Informatică pentru clasele IX-XII

TURBO PASCAL

ALGORITMI ŞI LIMBAJE DE PROGRAMARE

Manual pentru clasa a IX-a

Tudor Sorin

Teora

Conceput în conformitate cu programa școlară în vigoare, pentru disciplinele Algoritmi și limbaje de programare și Aplicații practice de laborator - clasa a IX-a, profil informatică - manualul este rezultatul îndelungatei experiențe didactice a autorului. Fiecare capitol este însoțit de un set de probleme propuse, prezentate în funcție de complexitatea lor. Elevilor li se oferă acum posibilitatea de a acoperi dintr-o singură sursă, atât programa cât și aplicații din afara ei, pentru care există un interes dovedit: unități de program, programare pe obiecte, crearea meniurilor, formatarea ecranelor, validarea datelor de intrare, reprezentări grafice de funcții și suprafețe, salvarea imaginilor etc.

Tudor Sorin

TURBO PASCAL

Algoritmi și limbaje de programare
Manual pentru clasa a IX-a

Teora

Acoperă programa cursurilor de algoritmi, limbaje de programare și lucrări practice de laborator.
Nu a făcut obiectul unei avizări a Ministerului Învățământului și Științei

Teora

CP 79-30, cod 72450 Bucureşti, România
Tel.: 619.30.04
Fax: 210.38.28

Distribuție

B-dul Al. I. Cuza nr. 39
Tel.: 222.45.33
Fax: 222.45.33

Teora – Cartea prin poștă

CP 79-30, cod 72450 Bucureşti, România
Tel.: 635.14.41

Copyright ©1995 **Teora**
Prima ediţie 1995
Retipărită: octombrie 1995
Reproducerea integrală sau parţială a textului sau a ilustraţiilor din această carte este posibilă numai cu acordul prealabil scris al editurii Teora.

Coperta: Valentin Tănase
ISBN: 973-601-212-3
Tipar executat la ROMFEL S.R.L.
Bucureşti, str. Lizeanu nr. 35 A, sect. 2

Tipărit la ROMFEL S.R.L.
Str. Lizeanu nr. 35A, sector 2

Cuprins

1. Noțiunea de algoritm, caracteristici 9
   1.1. Obiectele cu care lucrează algoritmii: date, variabile, expresii, operații 11
      1.1.1. Date 11
      1.1.2. Variabile 11
      1.1.3. Expresii 12
         1.1.3.1. Expresii întregi 12
         1.1.3.2. Expresii reale 12
         1.1.3.3. Expresii logice 13
         1.1.3.4. Expresii de tip șir de caractere 13
      1.1.4. Operații 13
         1.1.4.1. Operații de intrare-ieșire 13
         1.1.4.2. Operații de decizie 14
         1.1.4.3. Operații de atribuire 14

2. Principiile programării structurate 15
   2.1. Structuri de bază. Descrierea acestora cu ajutorul schemelor logice 15
      2.1.1. Operații de intrare-ieșire 15
      2.1.2. Operații de atribuire 16
      2.1.3. Operații de decizie 16
      2.1.4. Reguli de detaliere a unei operații complexe 18
      2.1.5. Structura alternativă 18
      2.1.6. Structura repetitivă 19
   2.2. Structuri de bază. Descrierea acestora cu ajutorul unui limbaj de tip pseudocod 22
      2.2.1. Prezentarea limbajului 22
         2.2.1.1. Variabile 22
         2.2.1.2. Instrucțiuni 23
   2.3. Aplicații 27

3. Elemente de bază ale limbajului Pascal 46
   3.1. Noțiuni introductive 46
      3.1.1. Evoluția limbajelor de programare 46
      3.1.2. Structura programelor Pascal 47
      3.1.3. Descrierea sintaxei cu ajutorul diagramelor de sintaxă . . . 48
   3.2. Vocabularul limbajului 50
      3.2.1. Setul de caractere 50
      3.2.2. Identificatori 50
      3.2.3. Separatori și comentarii 51
   3.3. Constante 51
      3.3.1. Constante întregi 51
      3.3.2. Constante reale 52
      3.3.3. Constante șir de caractere 52
      3.3.4. Constante simbolice 53
   3.4. Noțiunea de tip de dată 53
      3.4.1. Tipuri simple standard 54
         3.4.1.1. Tipul boolean 54
         3.4.1.2. Tipul char 54
         3.4.1.3. Tipuri întregi 54
         3.4.1.4. Tipuri reale 56

3.4.2. Tipuri ordinale definite de utilizator 57
3.4.2.1.Tipul enumerat 57
3.4.2.2. Tipul subdomeniu 58
3.4.3. Definirea tipurilor 58
3.5. Declararea variabilelor 58
3.6. Definirea constantelor 60
3.7. Expresii 60
3.8. Citirea și scrierea datelor 60
3.8.1. Citirea datelor 61
3.8.2. Scrierea datelor 61

4. Instrucțiunile limbajului Turbo Pascal 62
4.1. Instrucțiunea de atribuire 63
4.2. Instrucțiunea IF 64
4.2.1. IF THEN ELSE 64
4.2.2. Forma IF THEN 65
4.3. Instrucțiunea compusă 66
4.4. Instrucțiunea vidă 67
4.5. Instrucțiunea CASE 67
4.6. Instrucțiunea WHILE 68
4.7. Instrucțiunea REPEAT 70
4.8. Instrucțiunea FOR 71
4.9. Aplicații la capitolele 3 și 4 76

5. Tipuri structurate de date 81
5.1. Tipul tablou 81
5.1.1. Tipul ARRAY 81
5.1.2. Tipul STRING 90
5.1.2.1. Funcția COPY 94
5.1.2.2. Funcția POS 95
5.1.2.3. Procedura DELETE 96
5.1.2.4. Procedura INSERT 96
5.1.2.5. Procedura STR 97
5.1.2.6. Procedura VAL 98
5.2. Tipul înregistrare (RECORD) 98
5.2.1. Tipul înregistrare fixă 98
5.2.2. Tipul înregistrare cu variante 104
5.3. Tipul de date mulțime 106
5.3.1. Constructori de tip mulțime 107
5.3.2. Operatori 108
5.3.3. Constante 108
5.3.4. Constante de tip mulțime 110
5.4. Aplicații la capitolul 5 111
5.4.1. Tipul ARRAY 111
5.4.2. Tipul STRING 118
5.4.3. Tipul RECORD 118
5.4.4. Tipul mulțime 119

6. Subprograme 122
6.1. Conceptul de subprogram 122
6.2. Domeniul de vizibilitate a identificatorilor 122
6.3. Proceduri 123
6.3.1. Declarare și apel 123
6.3.2. Parametri formali, parametri efectivi 129
6.3.2.1. Transmiterea parametrilor prin referință 130
6.3.2.2. Transmiterea parametrilor prin valoare 130

6.4. Funcții (declarare și apel) 133
6.4.1. Tipul funcției 134
6.5. Dezvoltare ascendentă, dezvoltare descendentă 135
6.5.1. Dezvoltarea ascendentă 135
6.5.2. Dezvoltarea descendentă 136
6.6. Unități de program 136
6.6.1. Forma generală și construcția unei unități de program . . 136
6.7. Proceduri și funcții predefinite 139
6.8. Parametri formali de tip procedură sau funcție 140
6.8.1. Tipul de date procedural 140
6.9. Programarea pe obiecte 143
6.9.1. Logica și mecanismul programării pe obiecte 143
6.9.2. Mecanismul de realizare a programării pe obiecte 144
6.9.2.1. Sintaxa tipului obiect și încapsularea 144
6.9.2.2. Moștenirea 147
6.9.2.3. Atribuirea în cazul variabilelor obiect 148
6.9.2.4. Proceduri cu parametri formali de tip obiect . . 149
6.9.2.5. Polimorfism 150
6.10. Aplicații la capitolul 6 154
6.10.1. Proceduri, funcții, unități de program, obiecte 154

7. Unitatea de program CRT 159
7.1. Memoria video 159
7.2. Ferestre 161
7.3. Alte proceduri și funcții specifice unității CRT 164
7.4. Aplicații 167
7.4.1. Unitatea de program UTIL 167
7.4.1.1. Procedura DESCHID_F 167
7.4.1.2. Procedura SCRIU_F 167
7.4.1.3. Procedura A_D_BARA 168
7.4.1.4. Procedura PLIMB_BARA_V 169
7.4.1.5. Procedura PLIMB_BARA_O 170
7.4.1.6. Procedura CURSOR 170
7.4.2. Fereastra obiect 174
7.4.2.1. Metoda DESCHID 174
7.4.2.2. Metoda SALVEZ 175
7.4.2.3. Metoda RESTAUREZ 175
7.4.3. Meniul obiect 177
7.5. Aplicații la capitolul 7 182
7.5.1. Aplicații ale unității de program CRT 182

8. Fișiere Pascal 187
8.1. Fișiere text 187
8.1.1. Crearea și exploatarea fișierelor text 188
8.1.2. Citirea variabilelor de tip char 191
8.1.3. Citirea variabilelor de tip string 192
8.1.4. Citirea variabilelor de tip numeric 192
8.1.5. Scrierea cu format implicit 193
8.1.6. Scrierea cu format explicit 193
8.1.6.1. Scrierea datelor de tip întreg 193
8.1.6.2. Scrierea datelor de tip real 194
8.1.6.3. Scrierea datelor de tip caracter 194
8.1.6.4. Scrierea datelor de tip string 194
8.1.6.5. Scrierea datelor de tip boolean 194
8.1.7. Alte proceduri și funcții care lucrează cu fișiere text . . . 195
8.1.7.1. Procedura APPEND 195

8.1.7.1. Funcția EOLN 195
8.1.8. Fișiere text standard 196
8.1.8.1. Fișierele INPUT și OUTPUT 196
8.1.8.2. Fișierul PRN 196
8.2. Validarea operațiilor de intrare / ieșire 196
8.2.1. Alte tipuri de validări 198
8.2.1.1. Validarea naturii datelor 198
8.2.1.2. Testarea naturii numerice a datelor 199
8.2.1.3. Testarea naturii alfabetice a datelor 199
8.3. Fișiere cu tip 200
8.3.1. Crearea fișierelor cu tip 201
8.3.2. Procedura SEEK și funcția FILESIZE 204
8.3.3. Adăugarea de articole 205
8.4. Un exemplu de lucru cu un fișier 210
8.5. Fișiere fără tip 223
8.6. Aplicații la capitolul 8 225
8.6.1. Aplicații ale fișierelor 225
8.6.1.1. Fișiere text 225
8.6.1.2. Fișiere cu tip 226
8.6.1.3. Fișiere fără tip 228

9. Unitatea de program DOS 230
9.1. Prelucrarea datei 230
9.2. Căutarea unui fișier 231
9.3. Execuția unui program 231
9.4. Probleme propuse 234

10. Grafică pe calculator 235
10.1. Introducere 235
10.2. Inițializarea modului grafic 235
10.3. Culori 237
10.4. Coordonate ecran și reprezentarea punctelor 242
10.5. Punct curent 243
10.6. Trasarea segmentelor 243
10.7. Raportul aspect 244
10.8. Desenarea obiectelor grafice 245
10.8.1. Desenarea unei linii frânte 245
10.8.2. Desenarea cercurilor, arcelor de cerc și de elipsă 246
10.8.3. Desenarea unor obiecte hașurate 247
10.9. Afișarea textelor 250
10.10. Tehnici de animație 254
10.11. Fișiere imagine 260
10.12. Ferestre grafice 262
10.13. Elemente de grafică 2D 265
10.13.1. Desenarea graficului unei funcții 265
10.13.2. Desenarea curbelor plane 271
10.13.3. Rotații 275
10.13.4. Curbe BSPLINE 277
10.14. Elemente de grafică 3D 278
10.14.1. Rotația unei figuri în spațiu 285
10.14.2. Reprezentarea unei suprafețe în spațiu 287
10.14.3. Coordonatele sferice și aplicațiile lor 291
10.14.4. Prezentarea integrală a unității UTILG 295
10.15. Probleme propuse 297

ANEXA Utilizarea meniurilor în limbajul TURBO PASCAL 299

Capitolul 1

Noțiunea de algoritm, caracteristici

Să presupunem că dispunem de un calculator electronic de buzunar, fără program. Se cere să se calculeze valorile funcției de mai jos, definită pe mulțimea numerelor reale, pentru 10 valori date ale lui x (−3.1, 7, 0, 2.23, ș.a.m.d.).

$$f(x) \begin{cases} x*x-2, & x<0 \\ 3, & x=0 \\ x+2, & x>0 \end{cases}$$

Ce avem de făcut? Considerăm prima valoare a lui x, și anume −3.1. Observăm că această valoare este mai mică decât 0. Avem de calculat x*x−3. Pentru a face acest calcul îl introducem pe x, îl ridicăm la puterea a doua (am presupus că acest calculator poate efectua direct ridicarea la puterea a doua), din rezultat scădem 2 și obținem rezultatul dorit. Repetăm calculul pentru următoarele 9 valori. Să nu uităm faptul că pentru fiecare valoare a lui x, noi trebuie să decidem care din cele trei variante de calcul va fi aleasă. Să recunoaștem că o astfel de modalitate de lucru este plicticoasă și cere mult timp. Trebuie găsit un mod de lucru eficient. Care este acesta? Răspunsul la întrebare constituie subiectul actualului capitol.

Presupunem că dispunem de un calculator programabil. Trebuie să respectăm următoarele etape de lucru:

- elaborarea unui algoritm de rezolvare a problemei;
- transpunerea acestuia într-un limbaj de programare;
- rularea programului și obținerea rezultatelor.

Ce este un algoritm și cum se elaborează?

În primul rând, trebuie să ne fie foarte clar de la ce pornim și ce avem de făcut. Cunoaștem funcția și 10 valori ale lui x. Trebuie calculată valoarea funcției pentru fiecare din cele 10 valori. Sarcina noastră poate fi sistematizată astfel:

– se parcurg următoarele etape, de zece ori :
- se citește valoarea lui x;
- în funcție de valoarea propriu-zisă a lui x, se procedează astfel:
- dacă este negativă, se calculează f = x*x−2;
- dacă este 0, valoarea funcției este 3;
- dacă este pozitivă, se calculează f = x + 2;
- se scrie valoarea calculată pentru f.

Această sinteză a ceea ce avem de făcut se numește **algoritmul** de rezolvare a problemei propuse.

Noțiunea de algoritm este primară, nu se definește, întocmai ca și noțiunea de mulțime în matematică. Cu toate acestea, ca și mulțimea, algoritmul poate fi descris. În esență, este vorba de o succesiune de etape care se pot aplica mecanic pentru rezolvarea unei clase de probleme (pornind de la datele inițiale să conducă la soluție).

În exemplul anterior, algoritmul poate fi aplicat pentru oricare 10 valori ale lui x și nu numai pentru cele 10 date inițial. Dacă vom considera că pentru fiecare 10 valori avem o problemă, înseamnă că algoritmul nostru este capabil să rezolve o infinitate de probleme.

Pentru redactarea algoritmilor se folosesc **schemele logice** și **limbajele de tip pseudocod**. Despre acestea vom discuta în capitolul următor.

Un algoritm bun trebuie să satisfacă următoarele cerințe: claritate, generalitate și finititudine.

Claritatea este acea proprietate a algoritmului prin care procesul de calcul este descris precis, fără ambiguități.

Generalitatea este proprietatea algoritmului de a rezolva o întreagă clasă de probleme.

Revenind la problema prezentată la începutul acestui capitol, se observă că algoritmul poate fi ușor modificat pentru a calcula valoarea funcției într-un număr oarecare de puncte și nu numai în cele 10 presupuse inițial. Mai general, se poate obține un algoritm care să primească drept date de intrare atât numărul de puncte cu valorile lor, cât și funcția ale cărei valori trebuie calculate.

Finititudinea este proprietatea algoritmilor de a furniza rezultate într-un timp finit.

Referitor la această ultimă proprietate, se impun anumite precizări. Există probleme pentru care nu se cunosc algoritmi de rezolvare rapizi. Pentru anumite seturi de date de intrare, este necesar un timp de calcul de ordinul sutelor sau chiar miilor de ani și aceasta pe calculatoare ultramoderne. Din acest motiv, în practică se consideră că pentru aceste probleme nu există algoritmi de rezolvare, chiar dacă timpul de lucru este finit.

Observații:

1. Nu există un algoritm cu ajutorul căruia să putem elabora oricare alt algoritm. Acest fapt ne arată că activitatea de elaborare a algoritmilor nu este ușoară. Ea presupune multă muncă și fantezie.
2. O problemă poate avea mai mulți algoritmi de rezolvare. Unii dintre ei conduc mai rapid la rezultat. Este sarcina noastră să căutăm un algoritm care să rezolve rapid problema. Aceasta înseamnă că, atunci când avem de elaborat un algoritm, nu ne vom opri la prima soluție găsită.

Odată scris, un algoritm este codificat într-un limbaj de programare convenabil ales și astfel se obține **programul**. Acesta se rulează ori de câte ori este necesar. Există multe limbaje de programare, ca de exemplu FORTRAN, COBOL, C, PASCAL.

În acest manual vom studia limbajul PASCAL.

1.1. Obiectele cu care lucrează algoritmii: date, variabile, expresii, operații

1.1.1. Date

Așa cum am văzut, orice algoritm pornește de la anumite date de intrare, le prelucrează, iar în final obține date de ieșire. În exemplul prezentat, datele de intrare sunt reprezentate de cele 10 valori ale lui x, iar datele de ieșire sunt cele 10 valori ale lui f(x).

Datele pot fi clasificate după tipul lor:
- întregi;
- reale;
- logice;
- șir de caractere.

Datele din primele două tipuri poartă numele de *date numerice*.

Datele întregi sunt numere aparținând mulțimii numerelor întregi.

După cum știm din matematică, mulțimea numerelor reale este o reuniune între mulțimea numerelor raționale și mulțimea numerelor iraționale. Nici un calculator din lume nu poate reține un număr irațional, întrucât acesta are o infinitate de zecimale. Din acest motiv, în informatică, printr-o **dată reală** înțelegem un număr cu zecimale. La o astfel de dată, în loc de virgulă se folosește punctul.

Exemplu: în loc de 3,25 se scrie 3.25.

O **dată logică** poate avea numai două valori: **TRUE** (adevărat) sau **FALSE** (fals).

După cum indică și numele, o dată de tip **șir de caractere** înseamnă un șir de caractere cuprinse între două semne apostrof.

Exemplu: 'un text'.

Aici este momentul să vorbim de o categorie aparte de date și anume **constantele**. Acestea pot fi date de oricare dintre tipurile precizate. Ele se caracterizează prin faptul că sunt conținute în algoritm, fără a fi citite sau obținute din calcule.

Se folosesc în calculele care se fac sau sunt mesaje care trebuie să apară la fiecare rulare a programului rezultat în urma algoritmului.

1.1.2. Variabile

În problema analizată aveam de citit 10 valori de intrare. În matematică acestea ar fi fost notate x1, x2,...,x10. În prelucrare, pentru orice dată de intrare de acest tip se fac aceleași operații. Este lipsit de sens să explicăm ce facem cu x1 pentru ca apoi să explicăm ce facem cu x2 etc. Din acest motiv, în algoritm se explică ce se face cu un anume x, oricare ar fi el dintre cele 10. De asemenea, nu are rost să precizăm că se tipărește f(x1) apoi f(x2) ș.a.m.d. Se notează pur și simplu că

se tipărește f. Pentru algoritmul nostru, x și f sunt variabile. Orice algoritm lucrează cu variabile. Cu toate că o variabilă are un nume unic, conținutul ei poate fi diferit de la un moment la altul pe parcursul execuției programului. De aici provine și denumirea de variabilă. Dar dacă, în mod abstract, este posibil să notăm o intrare cu x și o ieșire cu f, cum este posibil să folosim această abstractizare în momentul în care algoritmul este transpus într-un limbaj de programare obținându-se un program ce urmează a fi rulat pe calculator? Practic, în memoria calculatorului se rezervă pentru fiecare variabilă câte un spațiu de memorie. Aceasta înseamnă că am rezervat un singur spațiu pentru x și un singur spațiu pentru f. Într-un astfel de spațiu se poate reține la un moment de timp o singură valoare. Tipurile de variabile coincid cu tipurile de date. Este normal să fie așa întrucât într-o variabilă se poate reține numai un anumit tip de date. În concluzie, printr-o variabilă înțelegem un ansamblu de patru elemente:

- numele variabilei;
- tipul ei;
- valoarea ei la un moment dat;
- locul în memoria calculatorului unde poate fi găsită variabila (adresa ei).

1.1.3. Expresii

Cu ajutorul constantelor, variabilelor și operatorilor se pot construi diverse **expresii**. Acestea se scriu după reguli precis determinate, dar (în general) aceste reguli sunt preluate din matematică. În scrierea expresiilor este permisă folosirea parantezelor.

Ca și variabilele, expresiile sunt de mai multe tipuri:

- întregi;
- reale;
- logice;
- de tip șir de caractere.

1.1.3.1. Expresii întregi

Folosesc operatorii +, −, * (pentru înmulțire), **DIV** (pentru câtul împărțirii întregi), **MOD** (pentru restul împărțirii întregi) ș.a.

Rezultatul evaluării lor este un număr întreg.

Exemple:

3*x+7, unde x este o variabilă întreagă;

a*(b+c), unde a,b,c sunt variabile întregi.

1.1.3.2. Expresii reale

Operanzii care le alcătuiesc sunt constante și variabile întregi sau reale. Pe lângă operatorii utilizați în cadrul expresiilor întregi putem utiliza și alții ca ar fi de exemplu / (pentru împărțirea cu zecimale). Rezultatul evaluării lor este întotdeauna un număr real.

1.1.3.3. Expresii logice

Rezultatul lor poate avea numai două valori: **TRUE** sau **FALSE**.

Operatorii sunt cei din logica matematică: **OR, AND, NOT**. De asemenea este permis să folosim și operatorii **<**, **>**, **>=** (mai mare sau egal), **<=** (mai mic sau egal), **<>** (diferit).

Exemple:
a OR b, unde a și b sunt variabile logice;
(a > = b)AND(c < d).

Pentru folosirea corectă a expresiilor logice trebuie să avem cunoștințe de logică matematică.

1.1.3.4. Expresii de tip șir de caractere

Se folosesc constante și variabile de tip șir de caractere. Singurul operator permis este operatorul **+** având semnificația de **concatenare** (scrierea a celor două șiruri unul după altul).

Exemplu: a+ ' text', unde a este o variabilă de tip șir de caractere având conținutul 'acest' are ca rezultat șirul 'acest text'.

1.1.4. Operații

În linii mari, operațiile care se fac într-un algoritm se pot clasifica în trei categorii:
- operații de intrare-ieșire;
- operații de atribuire;
- operații de decizie.

1.1.4.1. Operații de intrare-ieșire

Am arătat că orice algoritm lucrează cu **date de intrare** și **de ieșire**. Acestea pot fi citite sau scrise.

Prin **operația de intrare** (de citire) se înțelege preluarea unei date de la un dispozitiv de intrare către memoria internă a calculatorului, în spațiul rezervat pentru aceasta (spațiul rezervat pentru variabila care primește ca valoare acea dată).

Dispozitivele de intrare pot fi următoarele:
- tastatura (cel mai des utilizată);
- unitatea de dischetă;
- unitatea de disc.

Prin **operația de ieșire** (scriere) se înțelege trecerea unei date din memoria internă către un dispozitiv de ieșire.

Dispozitivele de ieșire pot fi următoarele:
- monitorul;
- unitatea de dischetă;
- unitatea de disc.

1.1.4.2. Operații de decizie

În cadrul unui algoritm, în funcție de datele de intrare se pot realiza anumite operații. În exemplul nostru, se face un anumit calcul atunci când data de intrare corespunzătoare variabilei x este negativă și altul în cazul că aceasta este pozitivă.

În funcție de valoarea datei de intrare, se ia o decizie sau alta. În general, în funcție de anumite **condiții**, trebuie făcute anumite operații sau altele. Operația prin care testăm acele condiții se numește **operație de decizie**. În funcție de rezultatul testului, algoritmul execută anumite operații.

1.1.4.3. Operații de atribuire

Am văzut ce este o variabilă. În algoritm aceasta poate lua diferite valori ca urmare a operațiilor de intrare sau a **operațiilor de atribuire**. Acest din urmă caz ne interesează acum.

Să presupunem că un algoritm folosește o variabilă notată arbitrar z și o alta notată y. Dorim ca z să ia o anumită valoare (de exemplu 1). Spunem că am atribuit variabilei z valoarea 1 și pentru moment vom nota acest lucru astfel: z:=1. Atât timp cât asupra variabilei z nu efectuăm nici o altă operație de atribuire și nu efectuăm nici o operație de citire, aceasta își păstrează valoarea 1. Presupunem că scriem valoarea variabilei z pe monitor. După scriere variabila z va avea în continuare valoarea 1. Presupunem că atribuim variabilei y valoarea variabilei z (y:=z). În urma atribuirii, valoarea variabilei y va fi 1 și valoarea variabilei z rămâne 1. Acest lucru este mai greu de înțeles la început. În imaginația noastră, dacă y a luat valoarea lui z ar trebui ca y să aibă valoarea 1 și z să nu mai aibă nici o valoare, ca și când cifra 1 s-a mutat efectiv în variabila y și ea nu se poate găsi simultan în două locuri.

Atribuim lui y valoarea z+2 (y:=z+2). Prin aceasta înțelegem că s-a efectuat z+2 (1+2) și y a luat valoarea 3. Valoarea conținută de variabila y înainte de această atribuire (1) se pierde iar y va avea valoarea 3. Atribuim variabilei x valoarea 1. Atribuim variabilei x valoarea x+x+3 (x:=x+x+3). În acest caz x va lua valoarea 5 (1+1+3). Vechea valoare 1 (a variabilei x) se pierde. Dacă repetăm această atribure (x:=x+x+3), x va lua valoarea 13 (5+5+3).

În concluzie, operația de atribuire se efectuează astfel:

- se evaluează expresia din partea dreaptă a operației de atribuire;
- valoarea care se obține astfel este preluată de variabila din partea stângă, iar valoarea avută de aceasta se pierde.

Întrebări recapitulative:

1. Dorim să calculăm suma 99+1. Concepem un procedeu de calcul astfel:
   - se pune 1 pe prima poziție;
   - se pun două cifre 0.

   Este acesta un algoritm?

2. Există un algoritm capabil să calculeze toate zecimalele numărului pi?

3. Care va fi valoarea variabilei x după secvența de atribuiri care urmează?
   x:=3; y:=1; x:=x+x; y:=x+y; x:=y.

Capitolul 2

Principiile programării structurate

Programarea structurată a apărut la începutul anilor 70. Este un concept cu o importanță fundamentală în scrierea algoritmilor. Cine învață să redacteze structurat algoritmii dispune de mari avantaje, cum ar fi:

- o mai mare ușurință de scriere a algoritmilor;
- odată scris, un algoritm se înțelege mult mai ușor (înainte de apariția programării structurate, după o perioadă, nici cel care redactase algoritmul nu îl mai înțelegea).

În principal, cel care redactează un algoritm utilizând programarea structurată respectă structurile care vor fi prezentate ulterior. În acest manual vom învăța să redactăm algoritmii direct structurat, varianta nestructurată ținând deja de istoria informaticii.

2.1. Structuri de bază. Descrierea acestora cu ajutorul schemelor logice

Pentru reprezentarea algoritmilor pot fi folosite **schemele logice**. Această reprezentare este utilă în special în perioada de inițiere. Orice schemă logică începe cu un bloc de **START** și se termină cu unul de **STOP**. Aceste două blocuri sunt unice. Ele se reprezintă așa cum se vede mai jos.

Figura 2.1.1.

Figura 2.1.2.

Am arătat că în orice algoritm se execută trei tipuri de operații: de intrare-ieșire, de atribuire și de decizie. Toate aceste operații se pot reprezenta cu ajutorul unor simboluri speciale.

2.1.1. Operații de intrare-ieșire

Aceste operații se reprezintă grafic cu ajutorul unor paralelograme cum sunt cele de mai jos:

Figura 2.1.3.

Figura 2.1.4.

2.1.2. Operații de atribuire

Pentru reprezentarea unei astfel de operații se folosește dreptunghiul:

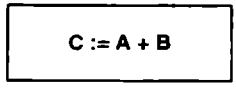


Figura 2.1.5.

2.1.3. Operații de decizie

Grafic, acestea se reprezintă utilizând rombul, așa cum rezultă din figura următoare:

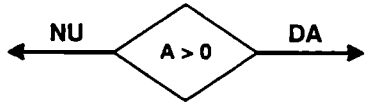


Figura 2.1.6.

În cazul algoritmilor cu un grad mai mare de complexitate, este dificil să scriem de la început o schemă logică completă și corectă. Din acest motiv, pentru acțiuni încă nedetaliate folosim un bloc special, cum este cel de mai jos, căruia îi atașăm un nume.

Figura 2.1.7.

Până acum, am arătat cum se reprezintă operațiile pe care le execută un algoritm. Trebuie arătată și **succesiunea** în care se execută aceste operații. Blocurile prezentate sunt unite între ele prin linii pe care se marchează sensul

de parcurgere (arce). În situația în care sensul de parcurgere este de sus în jos, marcarea nu mai este necesară.

Cum se reprezintă algoritmii cu ajutorul schemelor logice? Orice schemă logică conține obligatoriu cele două blocuri START și STOP, între care sunt reprezentate toate operațiile necesare prelucrării propriu-zise.

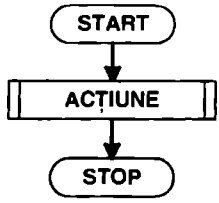


Figura 2.1.8.

Interpretarea acestei scheme este următoarea:

- se pornește de la blocul de START și se caută un drum către blocul de STOP;
- acest drum trece prin blocul **ACȚIUNE** (specific oricărui algoritm) care sintetizează **ce** trebuie să facem în algoritmul respectiv.

Urmează detalierea blocului ACȚIUNE. Într-o primă fază, se evidențiază una după alta operațiile algoritmului care pot fi citiri, scrieri, atribuiri, decizii sau alte operații complexe încă neprecizate în amănunt.

Exemplu: Să se citească un număr întreg (citirea se va face într-o variabilă a) și să se tipărească.

În acest caz, blocul ACȚIUNE poate fi detaliat astfel:

- citirea numărului;
- scrierea lui.

Schema logică corespunzătoare acestui algoritm este cea din figura următoare:

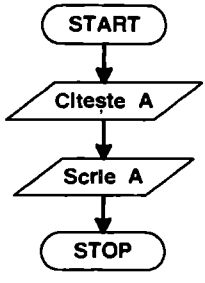


Figura 2.1.9.

Observații:

- structurile din figurile 2.1.8 și 2.1.9 sunt structuri **secvențiale**;
- forma generală a unei structuri secvențiale este cea din figura 2.1.10.

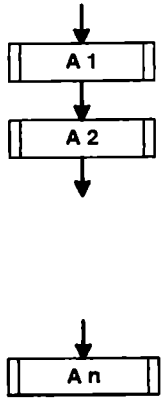


Figura 2.1.10.

2.1.4. Reguli de detaliere a unei operații complexe

Fiecare bloc ce desemnează o operație **complexă** (încă neprecizată în amănunt) se detaliază respectând următoarele structuri:
- structura secvențială (a fost prezentată);
- structura alternativă (cu cele două forme ale sale);
- structura repetitivă (cu cele două forme ale sale).

2.1.5. Structura alternativă

Cum intervin deciziile în schema logică? Modul de reprezentare grafică a unei decizii a fost deja arătat. Să presupunem un bloc ce reprezintă o operație complexă care constă din:
- evaluarea unei expresii logice;
- în cazul în care expresia ia valoarea TRUE, se execută o operație A1;
- în caz contrar (valoarea FALSE), se execută acțiunea A2.

Operațiile A1 și A2 pot fi operații de citire-scriere, atribuire sau alte operații ce necesită detaliere. În schemele logice, acestea se reprezintă printr-o structură ca aceea din figura 2.1.11., numită **structură alternativă** sau (pentru că deocamdată nu a fost studiată) **IF THEN ELSE**.

În anumite cazuri, după evaluarea expresiei logice, trebuie efectuate anumite operații numai dacă expresia ia valoarea TRUE. Se obține astfel o **structură alternativă de tip IF THEN**.

Exemplu de structură IF THEN ELSE: Se citesc două numere a și b. Să se tipărească cel mai mare dintre ele.

Analizăm ce avem de făcut:
- citim primul număr;
- citim al doilea număr;

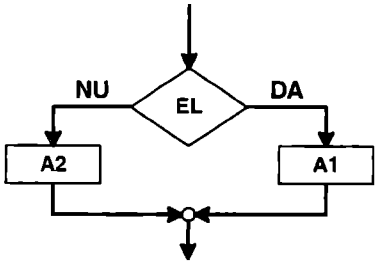


Figura 2.1.11.

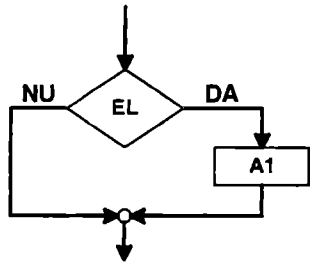


Figura 2.1.12.

- vedem care este mai mare și îl tipărim.

Ultima operație (complexă) trebuie transcrisă într-o structură alternativă. În mod practic, se procedează astfel:

- se verifică dacă a este mai mare decât b;
- dacă acest lucru este adevărat, se tipărește a;
- în caz contrar se tipărește b.

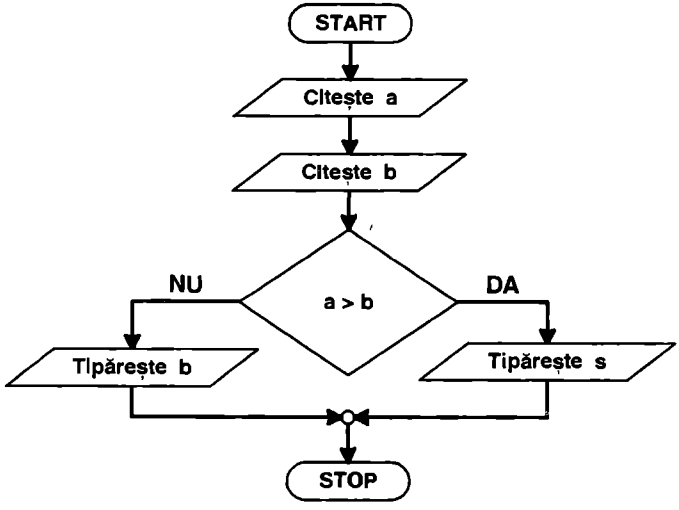


Figura 2.1.13.

2.1.6. Structura repetitivă

De multe ori, este necesară repetarea unor operații. Din acest motiv, în cadrul schemelor logice putem folosi **structura repetitivă**. Aceasta are două forme:

- structura repetitivă condiționată anterior;
- structura repetitivă condiționată posterior.

Structura repetitivă condiționată anterior se mai numește și **structura de tip WHILE DO**. Aceasta este folosită în cazul în care:

- avem de evaluat o expresie logică;
- dacă în urma evaluării obținem valoarea FALSE, se trece mai departe;
- dacă în urma evaluării obținem valoarea TRUE, se execută o operație (care poate fi și complexă), după care totul se reia. Reprezentarea grafică a acestei structuri se poate observa în figura următoare:

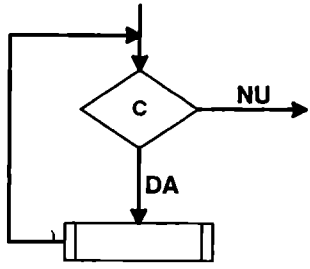


Figura 2.1.14.

Exemplu: Să presupunem că avem de calculat suma primelor n numere naturale. Mai jos este prezentată schema logică completă pentru această problemă.

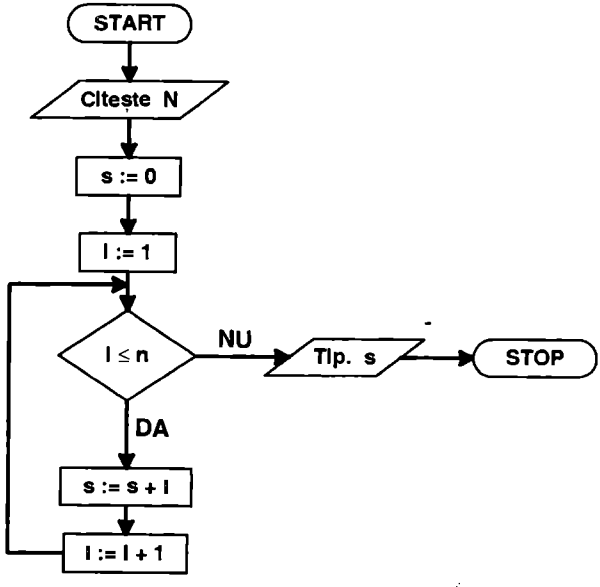


Figura 2.1.15.

Cum a fost alcătuită și cum se citește această schemă?

Trebuie să rezolvăm următoarele probleme:

- citirea lui n;

- inițializarea variabilei s, în care vom calcula suma (prin inițializare se înțelege că variabila capătă valoarea 0, în caz contrar este posibil ca variabila să aibă o altă valoare reziduală, deci rezultatul va fi greșit);
- să adunăm pe rând numerele 1,2,...,n la s;
  Pentru a realiza această ultimă operație procedăm astfel:
  - considerăm o variabilă i care va lua pe rând valorile 1,2,...,n (inițial i va lua valoarea 1);
  - de n ori îl însumăm pe i la s, de n ori îl mărim pe i cu 1 (controlul asupra faptului că facem această operație de n ori se realizează comparând pe i cu n).

Să probăm funcționarea algoritmului pe schema logică:
- se pornește de la blocul START;
- se citește n (pentru exemplificare presupunem valoarea 3);
- i va lua valoarea 1 iar s va lua valoarea 0;
- i este mai mic decât n (3), deci s va lua valoarea $0+1=1$ iar i va lua valoarea 2;
- i este mai mic decât n, deci s va lua valoarea $1+2=3$ iar i va lua valoarea 3;
- i este egal cu n, deci s va lua valoarea $3+3=6$ iar i va lua valoarea 4;
- i nu este mai mic sau egal cu n, deci se tipărește s (6) și algoritmul se oprește.

Schema repetitivă condiționată posterior se mai numește și **schema de tip REPEAT UNTIL**. Se folosește în cazul în care:
- avem de executat o anumită operație;
- evaluăm o expresie logică;
  - în cazul în care expresia ia valoarea TRUE, se trece mai departe;
  - în caz contrar, se reia totul.

Grafic, aceasta se reprezintă astfel:

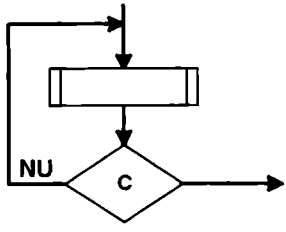


Figura 2.1.16.

Temă : Refaceți algoritmul anterior utilizând această structură.

Observație : Așa cum reiese și din cele două desene, poziția blocului de decizie este diferită față de cea a blocului de calcul. Astfel, în primul caz este posibil să nu se ajungă la blocul de calcul, în timp ce, în cazul al doilea se trece cel puțin o dată prin acesta.

2.2. Structuri de bază. Descrierea acestora cu ajutorul unui limbaj de tip pseudocod

Practica a demonstrat că utilizarea schemelor logice în programare este anevoioasă din următoarele motive:

- spațiul mare ocupat de acestea, fapt ce conduce la urmărirea dificilă a algoritmului;
- posibilități mici de inserare a comentariilor;
- neplăceri legate de reprezentarea grafică.

Toate acestea au determinat marea majoritate a programatorilor profesioniști să renunțe la ele. Trebuia găsit ceva care să le înlocuiască cu succes. Astfel, au apărut **limbajele de tip pseudocod**. Ce sunt ele? În primul rând sunt limbaje pentru descrierea algoritmilor. Ele conțin condensat, sub formă de **instrucțiuni**, subschemele clasice ale programării structurate. Cu ajutorul lor se pot descrie cu ușurință algoritmii. Există o mare asemănare între ele și limbajele de programare. Acest fapt permite ca, odată descris un algoritm într-un limbaj de tip pseudocod, conversia într-un limbaj de programare să se facă aproape în mod mecanic. Din acest motiv, mai toate cursurile referitoare la algoritmi care apar în străinătate și în țară folosesc pentru descrierea lor aceste limbaje. Se pune întrebarea următoare: dacă aceste limbaje de tip pseudocod sunt atât de asemănătoare limbajelor de programare, care este motivul pentru care algoritmii nu se descriu direct într-un limbaj de programare? Iată numai două din motivele care au condus la această situație:

- limbajele de tip pseudocod au o sintaxă mai liberă decât orice limbaj de programare;
- un algoritm scris într-un limbaj de tip pseudocod este înțeles de toți programatorii, în timp ce unul scris într-un anumit limbaj de programare va fi înțeles numai de cunoscătorii limbajului respectiv.

Există o mare diversitate de limbaje de tip pseudocod. Practic, oricine își poate concepe propriul limbaj. Totuși, în practică, s-a impus cel prezentat aici, datorită faptului că în străinătate au apărut multe lucrări privind algoritmii în care aceștia sunt descriși tocmai în acest limbaj.

Înaintea prezentării limbajului trebuie să înțelegem noțiunea de „**cuvânt cheie**" care intervine în studiul acestuia. Într-un limbaj există anumite cuvinte care au un înțeles strict. Ele nu pot fi folosite în nici un alt context.

2.2.1. Prezentarea limbajului

2.2.1.1. Variabile

Variabilele pot fi clasificate după tipul lor.

1. Variabile întregi

Au capacitatea de a reține numere întregi. Pentru recunoașterea lor se folosește cuvântul cheie **INTEGER**.

Exemplu : `INTEGER a,b.` Am declarat două variabile întregi a și b.

2. Variabile reale

Cu ajutorul lor se reține numere cu virgulă (raționale). Pentru recunoașterea lor folosim cuvântul cheie **REAL**.

Exemplu : `REAL c,d,e.` Am declarat trei variabile reale numite c, d și e.

3. Variabile logice (boolene)

Au capacitatea de a reține numai două valori și anume **TRUE** (adevărat) și **FALSE** (fals).

Exemplu : `BOOLEAN h;` Am declarat o variabilă numită h, de tip boolean.

2.2.1.2. Instrucțiuni

În capitolul anterior am arătat faptul că un algoritm execută trei tipuri de operații și anume: de intrare-ieșire, de atribuire și de decizie. Fiecărei operații, în limbaj îi corespunde o **instrucțiune.**

1. Instrucțiuni de intrare-ieșire

a. Instrucțiunea READ

Preia o informație din exterior în memoria principală. Pentru simplitate considerăm că informația este preluată de la tastatură.

Exemplu:
```
INTEGER a
REAL b
READ a,b
```

Se citesc două numere, unul întreg, altul real. După citire cele două numere se găsesc în variabilele a și b.

b. Instrucțiunea WRITE

Preia o informație din memorie (din cadrul unei variabile) și o scrie în exterior. Pentru simplitate, considerăm faptul că informația este scrisă pe monitor.

Exemplu:
```
INTEGER a
READ a
WRITE a.
```

Se citește de la tastatură un număr întreg în cadrul variabilei a (pe scurt putem spune că citim un număr a). Acest număr se tipărește pe monitor.

2. Instrucțiunea de atribuire

Este de forma v:=e.

Semnificația este următoarea:
- v - nume de variabilă;

- e - o expresie.

În acest caz trebuie ca tipul expresiei să coincidă cu tipul variabilei.

Exemple:
```
INTEGER a;
a:=2
a:=a+3.
```

În urma execuției acestei secvențe a va conține numărul întreg 5.

```
INTEGER a,b
BOOLEAN c;
a:=3
b:=4
c:=a>b.
```

În urma execuției acestei secvențe, c va avea valoarea FALSE.

3. Instrucțiunea STOP

Are rolul de a marca sfârșitul unui program.

4. Instrucțiuni de decizie

Aceste instrucțiuni se împart în două categorii:
- instrucțiuni de ramificare;
- instrucțiuni repetitive.

a. Instrucțiunea de ramificare

Corespunde structurii alternative de la scheme logice. Forma ei este următoarea:

```
IF conditie  THEN s1
             [ELSE s2]
ENDIF
```

Aici, s1, s2 reprezintă secvențe de alte instrucțiuni.

Ceea ce este încadrat între paranteze drepte este opțional.

Principiul de funcționare al acestei instrucțiuni (în cazul formei complete) este următorul:
- se evaluează condiția;
  - dacă aceasta este îndeplinită se execută secvența s1;
  - în caz contrar se execută secvența s2.
- se trece la instrucțiunea următoare.

În cazul în care clauza ELSE lipsește, principiul de funcționare este:
- se evaluează condiția;
- dacă aceasta este îndeplinită, se execută secvența de instrucțiuni s1;
- în caz contrar, se trece mai departe.

Exemplu de program:

Se citesc două variabile întregi. Să se tipărească cea mai mare dintre ele.

```
INTEGER a,b;
READ a,b
IF a>b   THEN WRITE a
         ELSE WRITE b
ENDIF
STOP.
```

b. Instrucțiunea repetitivă (WHILE)

Forma generală este:

```
WHILE expresie logica
      s
REPEAT.
```

s reprezintă o secvență de instrucțiuni.

Principiul de funcționare este următorul:

- se evaluează expresia logică;
- dacă aceasta are valoarea FALSE se trece la instrucțiunea următoare;
- dacă valoarea ei este TRUE, se execută secvența s după care se pornește din nou cu evaluarea expresiei logice.

Procedeul continuă până când, la evaluare, expresia logică are valoarea FALSE. Această instrucțiune corespunde în limbajul schemelor logice structurii repetitive condiționate anterior.

Să se scrie un program care calculează suma primelor n numere naturale (se presupune că nu se cunoaște formula directă).

```
INTEGER n,i,s;
READ n
s:=0
i:=1
WHILE i<=n
s:=s+i
i:=i+1
REPEAT
WRITE s.
```

Să analizăm funcționarea pentru n = 3:

- se citește n (citesc 3);
- s ia valoarea 0 iar i ia valoarea 1;
- i este mai mic decât n (1 este mai mic decât 3) deci s va lua valoarea 0 + 1 = 1 iar i va lua valoarea 2;
- i este mai mic decât trei deci s va lua valoarea 1 + 2 iar i va lua valoarea 3;
- i este egal cu n (deci este îndeplinită condiția) caz în care s va lua valoarea 3 + 3 = 6 iar i va lua valoarea 4;
- i este mai mare decât n, deci se trece la instrucțiunea următoare;
- se tipărește 6.

Instrucțiunile prezentate până aici sunt suficiente pentru descrierea unui algoritm. În practică se folosesc și alte două instrucțiuni repetitive care determină ca programul să fie mult mai ușor de citit.

5. Instrucțiunea repetitivă DO

Corespunde structurii repetitive condiționate posterior. Forma ei este următoarea:

```
DO
  s
UNTIL expresie logica.
```

Principiul de funcționare este următorul:

- se execută secvența *s*;
- se evaluează expresia logică;
- dacă în urma evaluării expresiei se obține valoarea TRUE, se trece la următoarea instrucțiune;
- în cazul în care expresia ia valoarea FALSE, se reia execuția secvenței s ș.a.m.d. Secvența s este executată până când expresia logică ia valoarea TRUE.

Pentru exemplificare, reluăm problema anterioară:

```
INTEGER n,i,s;
READ n
i:=1
s:=0
DO
s:=s+i
i:=i+1
UNTIL i>n
WRITE s
STOP
```

Să vedem cum decurge algoritmul după citirea lui n (citim 3):

- i ia valoarea 1 iar s ia valoarea 0;
- s ia valoarea 0+1=1 iar i ia valoarea 2;
- i nu este mai mare decât 2, deci s va lua valoarea 1+2=3 iar i va lua valoarea 3;
- i nu este mai mare ca 3 deci s va lua valoarea 3+3=6 iar i va lua valoarea 4;
- i este mai mare decât *n* deci se trece la instrucțiunea următoare unde se tipărește valoarea 6.

6. Instrucțiunea repetitivă FOR

Forma generală a acestei instrucțiuni este următoarea:

```
FOR variabilă de ciclare:= valoare inițială, valoare finală
   s
REPEAT
```

Variabila de ciclare trebuie să fie de tipul INTEGER. Valoare inițială, valoare finală sunt expresii aritmetice de tipul INTEGER.

Principiul de funcționare este următorul:

- variabilei de ciclare i se atribuie valoarea inițială;
- se execută secvența s;
- variabilei de ciclare i se adaugă 1;
- dacă nu s-a depășit valoarea finală se execută secvența s;
- procedeul continuă până când variabila de ciclare depășește valoarea finală. În situația în care, de la bun început valoarea inițială depășește valoarea finală, se trece la instrucțiunea următoare.

Reluăm problema referitoare la calculul sumei primelor n numere naturale:

```
INTEGER n,s,i;
READ n
s:=0
FOR i:=1,n
```

```
s:=s+i
REPEAT
WRITE s
STOP.
```

Algoritmul decurge astfel:

- se citește n (presupunem n = 3);
- s ia valoarea 0, iar i valoarea 1;
- se calculează s = 0 + 1 = 1;
- i ia valoarea 2;
- se calculează s = 1 + 2 = 3;
- i ia valoarea 3;
- se calculează s = 3 + 3 = 6;
- i ia valoarea 4 și în acest fel se depășește valoarea lui *n*;
- se trece la instrucțiunea următoare unde se tipărește valoarea 6.

2.3. Aplicații

În acest paragraf se prezintă câteva probleme rezolvate. Algoritmul propus pentru ele este prezentat atât sub formă de schemă logică cât și în limbaj pseudocod.

Problema 1

Se citesc două variabile de tip întreg, A și B. Să se interschimbe conținutul celor două variabile și să se tipărească.

La prima vedere, am fi tentați să scriem secvența următoare:

```
INTEGER a,b;
READ a,b
a:=b
b:=a
WRITE a,b
STOP.
```

Această secvență este greșită. Atunci când variabilei a i se atribuie valoarea variabilei b, conținutul variabilei a se pierde. Să presupunem că am citit a = 3 și b = 4.

Prin atribuirea a:=b, conținutul lui a va fi 4 iar al lui b va fi tot 4. Prin atribuirea b:=a conținutul celor două variabile va fi același. Din acest motiv, în urma execuției acestui algoritm se va tipări 4 și 4. Pentru a lucra corect, trebuie considerată o variabilă intermediară c de același tip ca a și b. Secvența corectă este următoarea:

```
c:=a
a:=b
b:=c
```

În acest mod, conținutul variabilei a se salvează în c (deci nu se pierde la a doua atribuire). Pentru valorile 3 și 4 presupuse ca citite pentru a și b vom avea în ordine:

- lui c i se atribuie valoarea 3;

- lui a i se atribuie valoarea 4;
- lui b i se atribuie valoarea 3.

Prezentăm mai jos algoritmul sub formă de schemă logică și pseudocod.

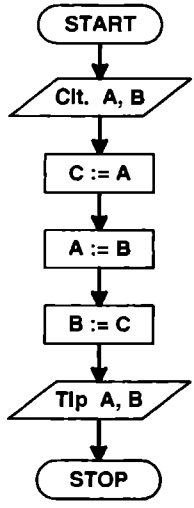


Figura 2.3.1.

```
INTEGER a,b,c;
READ a,b
c:=a
a:=b
b:=c
WRITE a,b
STOP.
```

În anumite cazuri se poate evita folosirea unei variabile intermediare. Să considerăm algoritmul următor realizat în pseudocod. Acesta rezolvă aceeași problemă.

```
INTEGER a,b;
READ a,b
a:=a+b
b:=a-b
a:=a-b
WRITE a,b
STOP.
```

Pentru valorile citite ale lui a și b algoritmul funcționează astfel:
- lui a i se atribuie valoarea $3+4=7$;
- lui b i se atribuie valoarea $7-4=3$;
- lui a i se atribuie valoarea $7-3=4$;
- se tipărește $a=4$ și $b=3$ (rezultat corect).

Problema 2

Să se elaboreze algoritmul pentru rezolvarea unei ecuații de gradul 1.

Forma generală a ecuației de gradul 1 este $a*x+b=0$. În matematică se presupune că a este diferit de 0. În programare acest lucru nu este posibil întrucât, din greșeală, cel care îl introduce pe a îi poate da acestuia valoarea 0. Din acest motiv se va lua în discuție și acest caz. Din analiza problemei rezultă că a poate avea valoarea 0 sau valori diferite de 0. În cazul în care a este diferit de 0 se calculează soluția $x:=-b/a$. Dacă a este 0, trebuie să vedem cum este b. Dacă și b este 0, înseamnă că orice număr real verifică ecuația, în caz contrar ecuația nu are soluție.

Trebuie să citim variabilele *a* și *b*, să le prelucrăm dând soluția sau mesajele corespunzătoare. O primă schemă logică ar arăta astfel:

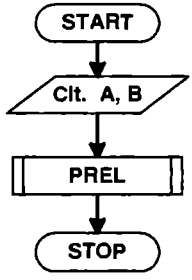


Figura 2.3.2.

Analizăm ce avem de făcut în operația PREL. Vom proceda în mod diferit pentru cazurile a = 0 și pentru a diferit de 0. Dacă a este diferit de 0, putem calcula x și să-l tipărim. Cazul în care a=0 nu-l analizăm pe moment. Schema devine:

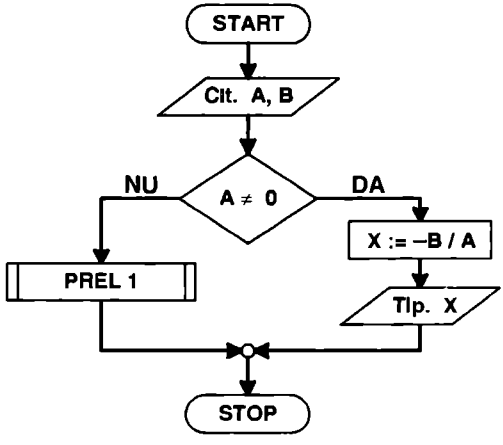


Figura 2.3.3.

Analizăm PREL1. Acum știm că a este 0. Procedăm diferit în funcție de valorile lui b. Dacă b este 0, avem o infinitate de soluții, în caz contrar nu avem nici una. Rezultă schema finală:

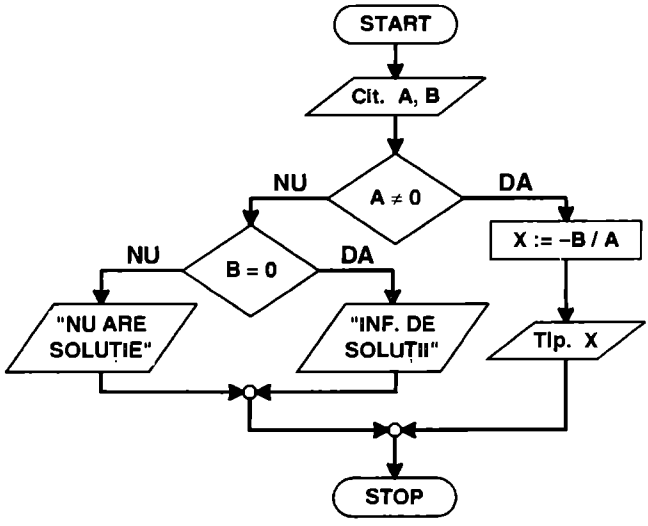


Figura 2.3.4.

În pseudocod, algoritmul arată astfel:

```
REAL a,b,x;
READ a,b
IF a<>0 THEN
  x:=-b/a
  WRITE x
ELSE
  IF b=0   THEN WRITE 'infinitate de solutii'
           ELSE WRITE 'nu avem solutii'
  ENDIF
ENDIF
STOP.
```

Problema 3

Se dau trei numere reale a, b, c.
Să se calculeze valoarea expresiei:

$$e = \begin{cases} a \cdot b, \text{ dacă } c>0 \\ ab, \text{ dacă } c=0 \\ a-b, \text{ dacă } c<0 \end{cases}$$

După ce citim a, b, c vedem dacă *c* este mai mare decât 0. În caz afirmativ, calculăm a + b. În caz contrar se efectuează operația PREL, pe care o vom detalia ulterior. O primă formă a schemei logice se găsește mai jos.

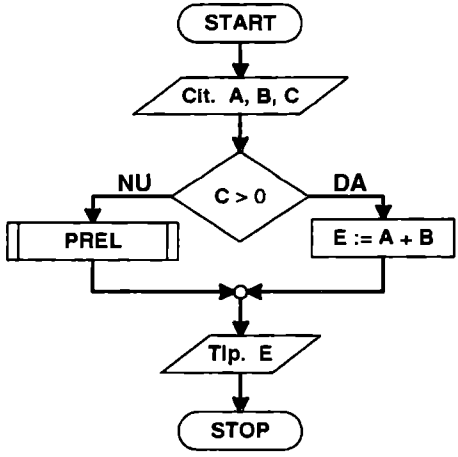


Figura 2.3.5.

Dacă c nu este mai mare decât 0, înseamnă că este mai mic sau egal cu 0. Testăm dacă el nu este 0. În caz afirmativ calculăm produsul a*b, în caz contrar calculăm a–b. Acum putem scrie schema logică finală.

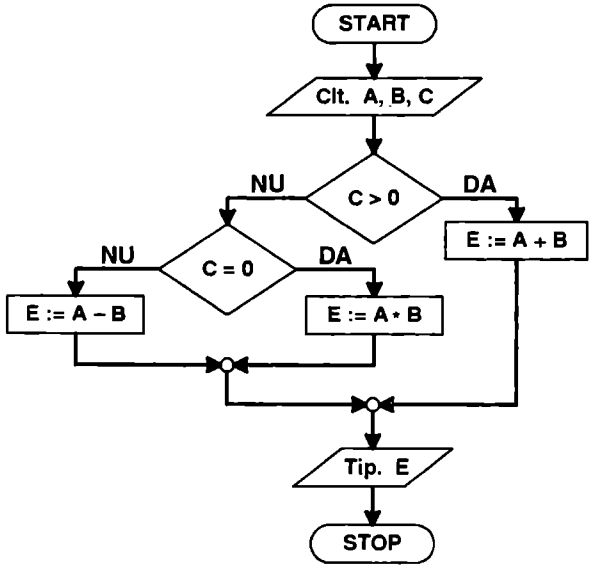


Figura 2.3.6.

În pseudocod algoritmul arată astfel:

```
REAL a,b,c,e;
READ a,b,c
IF c>0 THEN
  e:=a+b
```

```
ELSE   IF c=0   THEN e:=a*b
                ELSE e:=a-b
       ENDIF
ENDIF
WRITE e
STOP.
```

Problema 4

Se citesc trei numere întregi a, b, c. Să se tipărească în ordine crescătoare.

Ideea de lucru este următoarea:

- se compară primele două (a și b);
- după ce se cunoaște ordinea între acestea, se încearcă plasarea lui c față de ele.

În prima formă, schema logică arată astfel:

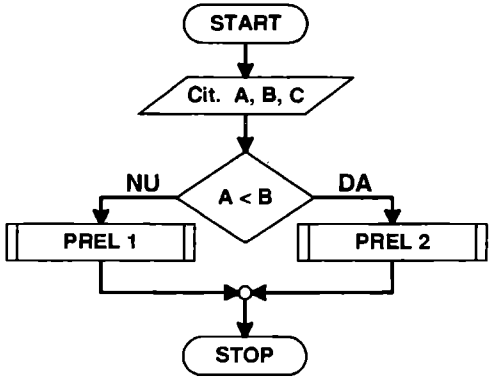


Figura 2.3.7.

Dacă a este mai mic decât b trebuie văzut unde îl plasăm pe c. El poate fi mai mic decât a, deci ordinea ar fi c, a, b, sau mai mare sau egal cu a. În acest din urmă caz, c trebuie comparat cu b. Dacă el este mai mic decât b, ordinea este a, c, b, în caz contrar – este a, b, c. Tot așa se raționează în situația în care a nu este mai mic decât b (este mai mare sau egal). În final se obține schema logică din figura 2.3.8. În pseudocod, algoritmul arată astfel:

```
INTEGER a,b,c;
READ a,b,c
IF a<b THEN
  IF c<a   THEN WRITE c,a,b
           ELSE  IF c<b   THEN   WRITE a,c,b
                          ELSE   WRITE a,b,c
                 ENDIF
  ENDIF
ELSE   IF c<b   THEN WRITE c,b,a
                ELSE  IF c<a   THEN   WRITE b,c,a
                                ELSE   WRITE b,a,c
                      ENDIF
       ENDIF
ENDIF
STOP.
```

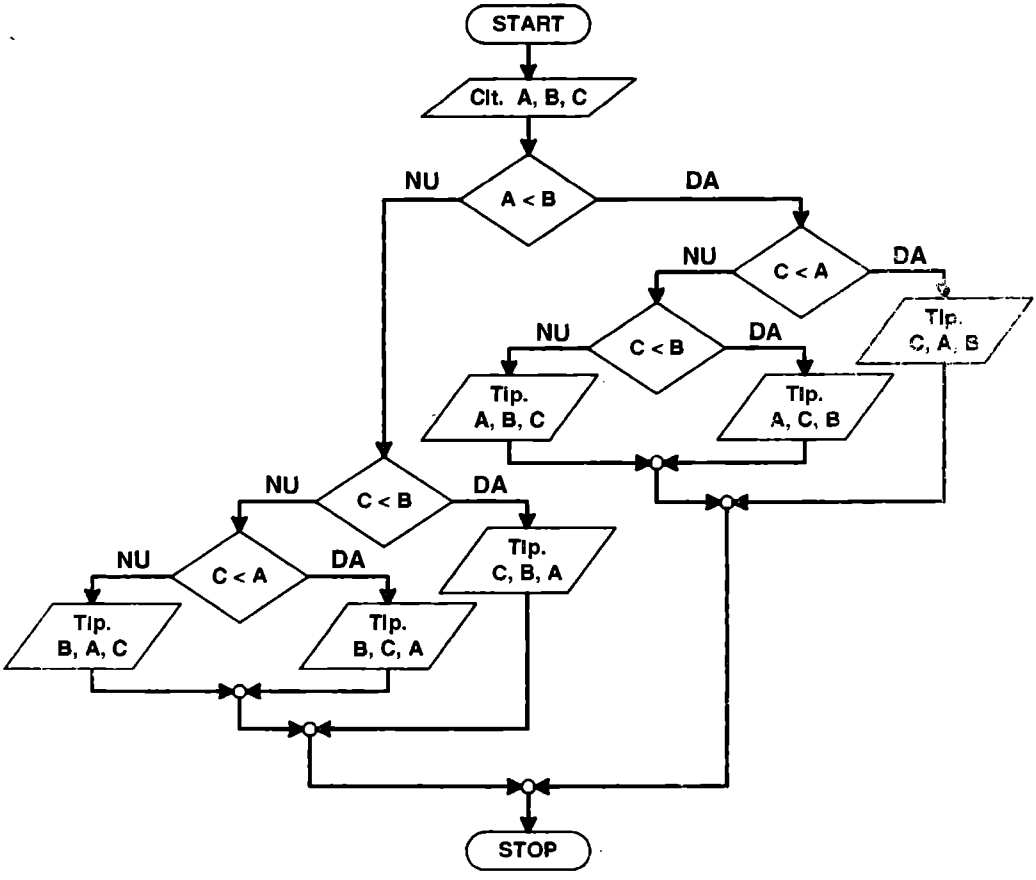


Figura 2.3.8.

O altă modalitate (mai generală) de rezolvare a acestei probleme va fi descrisă în continuare. Considerăm trei numere oarecare, spre exemplu 7, 4, 1. Avem dreptul de a inversa pozițiile a două numere alăturate. Dorim ca, în final, numerele să fie scrise în ordine crescătoare.

Iată cum procedăm:

- 7 este mai mare decât 4, deci se inversează cele două numere, obținând 4 7 1;
- 7 este mai mare decât 1, deci se inversează cele două numere 4 1 7;
- 4 este mai mare decât 1, deci obținem 1 4 7.

Acest exemplu numeric ne sugerează posibilitatea de a rezolva altfel problema anterioară:

- în locul numerelor considerăm variabilele a, b, c;
- comparăm pe rând a cu b, b cu c și din nou a cu b, iar dacă este cazul interschimbăm conținutul acestor variabile.

În pseudocod, algoritmul propus arată astfel:

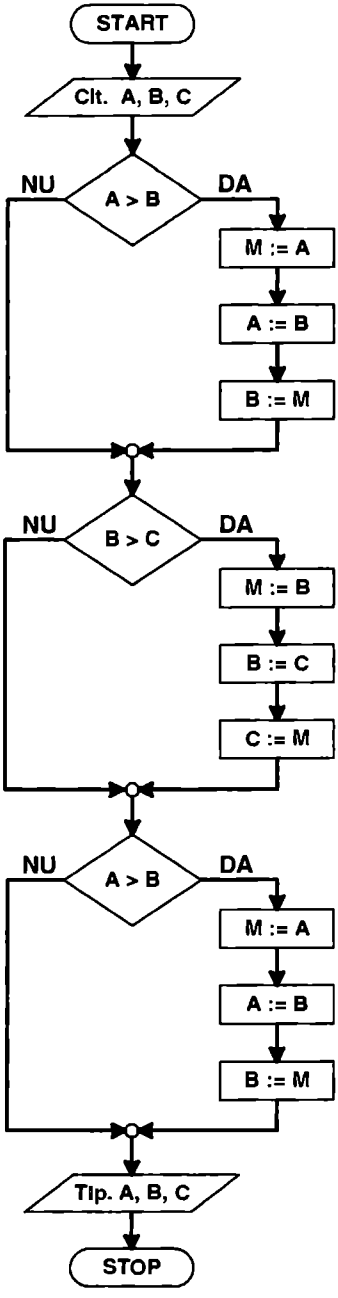


Figura 2.3.9.

```
INTEGER a,b,c;
READ a,b,c
IF a>b THEN
  m:=a
```

```
    a:=b
    b:=m
  ENDIF
  IF b>c THEN
    m:=b
    b:=c
    c:=m
  ENDIF
  IF a>b THEN
    m:=a
    a:=b
    b:=m
  ENDIF
  WRITE a,b,c
  STOP.
```

Problema 5

Pentru n citit (n natural mai mare sau egal cu 1) să se calculeze suma $S = 1 + 1*2 + 1*2*3 + \ldots + 1*2*\ldots*n$.

Pentru rezolvare, observăm că putem adăuga la o sumă s (care are inițial valoarea 0) pe rând valorile calculate $1, 1*2, \ldots, 1*2*\ldots*n$ (adică n valori). Aceasta nu înseamnă că trebuie calculată de fiecare dată valoarea $1*2*\ldots*i$. Dacă am calculat $1*2*\ldots*i$, valoarea următoare care trebuie adăugată este $1*2*\ldots*i*(i+1)$ și se poate obține înmulțind valoarea calculată cu $i+1$. Pentru calculul acestor valori se utilizează variabila p. Din cele arătate rezultă schema logică de mai jos:

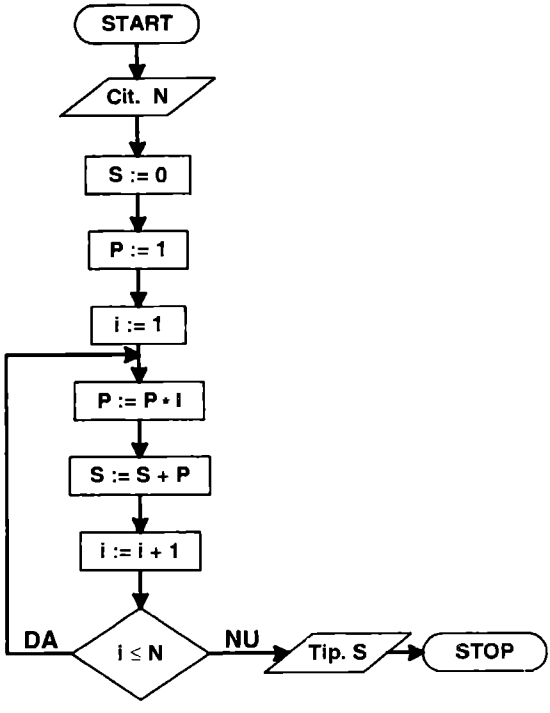


Figura 2.3.10.

În pseudocod, algoritmul propus arată astfel:

```
INTEGER n,i,s,p;
READ n
s:=0
p:=1
FOR i:=1,n
  p:=p*i
  s:=s+p
REPEAT
WRITE s
STOP.
```

Problema 6

Se citește un număr natural n. Să se calculeze suma cifrelor sale.

Exemplu: pentru n = 213, se va tipări 6 (2 + 1 + 3).

Pentru calculul acestei sume, trebuie să găsim o modalitate de a separa cifrele unui număr. Între operatorii care se pot aplica numerelor naturale sunt doi care ne folosesc în acest caz. Aceștia sunt operatorii DIV și MOD. Aplicând operatorul DIV pentru două numere naturale a și b sub forma *a DIV b* obținem câtul întreg al împărțirii lui a la b.

Exemplu: 24 DIV 5 = 4.

Prin aplicarea operatorului MOD la două numere naturale a și b (a MOD b) obținem restul împărțirii lui a la b.

Exemplu: 24 MOD 7 = 3.

Dacă vom considera un număr natural n, n MOD 10 va da restul împărțirii la 10 a numărului, deci ultima cifră a sa, iar n DIV 10 va avea ca rezultat câtul împărțirii la 10 a numărului (acesta nu este altceva decât numărul format prin suprimarea ultimei cifre a numărului n).

Presupunem că am citit numărul 425. Procedăm astfel:

- inițializăm suma (s = 0);
- calculăm ultima cifră a numărului 425 MOD 10 = 5 și o adunăm la s (s = 5);
- reținem numărul fără ultima cifră (425 DIV 10 = 42);
- pentru numărul nou format reținem ultima cifră și o adunăm la s (42 MOD 10 = 2, s: = 5 + 2 = 7);
- reținem numărul fără ultima cifră (42 DIV 10 = 4);
- pentru numărul nou format reținem ultima cifră și o adunăm la s (4 MOD 10 = 4, s: = 7 + 4 = 11);
- reținem numărul fără ultima cifră (4 DIV 10 = 0).

Întrucât numărul a devenit 0, algoritmul se încheie prin tipărirea lui s (s = 11).

Alăturăm algoritmul scris în pseudocod:

```
INTEGER n,s;
READ n
s:=0;
WHILE n<>0
```

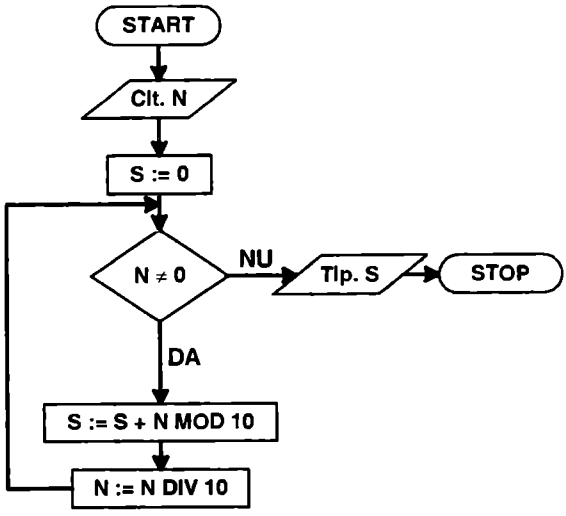


Figura 2.3.11.

```
  s:=s+n MOD 10
  n:=n DIV 10
REPEAT
WRITE s;
STOP.
```

Problema 7

Se citește n număr natural. Să se tipărească numărul obținut prin inversarea cifrelor lui n.

Exemplu: n=412. Se va tipări 214.

În problema anterioară am văzut cum putem izola cifrele unui număr. De menționat că algoritmul propus anterior izola cifrele numărului în ordine inversă.

Exemplu: pentru numărul 162 se izolau pe rând cifrele 2, 6 și 1.

Pe de altă parte, dacă cunoaștem cifrele unui număr putem calcula numărul respectiv.

Exemplu: din cifrele 2, 6 și 1 putem obține numărul 261. Aici nu este vorba de alăturarea cifrelor, cum am fi tentați să credem, ci de calculul numărului sub forma:

$$2*10^2+6*10^1+1*10^0,$$

așa cum cunoaștem din teoria bazelor de numerație. Cum procedăm pentru rezolvarea acestei probleme? Considerăm o variabilă (notată ninv, care inițial va avea valoarea 0). Pentru fiecare cifră noua valoare a lui ninv va fi vechea valoare înmulțită cu 10 la care se adaugă cifra:

ninv:=0*10+2=2;
ninv:=2*10+6=26;
ninv:=26*10+1=261.

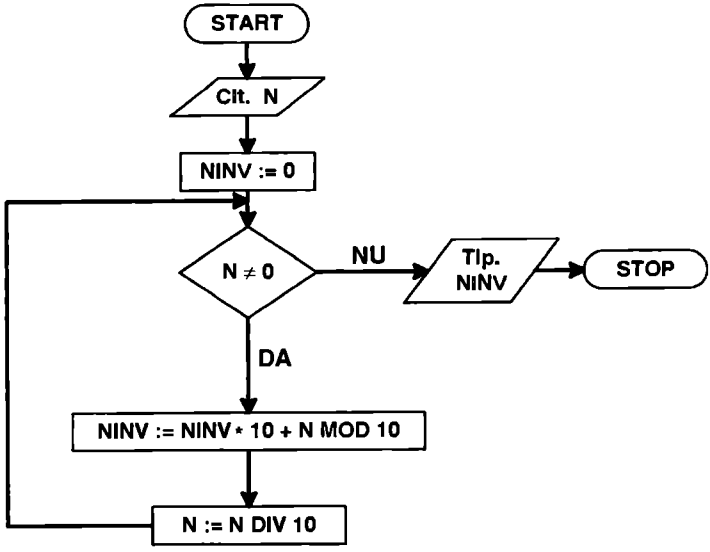


Figura 2.3.12.

În concluzie, izolăm pe rând cifrele în ordine inversă (așa cum am arătat în problema anterioară) și cu fiecare nouă cifră construim numărul cerut.

În pseudocod, algoritmul arată astfel:

```
INTEGER n,ninv;
READ n
ninv:=0;
WHILE n<>0
  ninv:=ninv*10+n MOD 10;
  n:=n DIV 10
REPEAT
WRITE ninv
STOP.
```

Problema 8

Se citește n număr natural. Să se precizeze dacă este prim sau nu.

Pentru rezolvare, o primă idee de lucru constă în a considera toți divizorii posibili ai numărului n. Dacă nici unul din aceștia nu-l divide, înseamnă că numărul este prim. Dar care sunt acești divizori posibili?

- o primă posibilitate ar fi să-i considerăm cuprinși între 2 și n–1;
- o a doua posibilitate este de a-i considera cuprinși între 2 și jumătatea lui n (n DIV 2) soluție mult mai avantajoasă decât prima întrucât considerăm mai puțini divizori și implicit se calculează mai puțin;
- o a treia posibilitate constă în a considera toți divizorii cuprinși între 2 și parte întreagă din radical din n (presupunând cunoscută teorema care afirmă că, dacă un număr nu are nici un astfel de divizor, el este prim).

Ultima variantă este cea mai bună. Spre exemplificare, vom considera

numărul 999; în prima variantă trebuie verificați 997 de divizori, în a doua 447 iar în a treia numai 29 de divizori. Corespunzător acestei ultime variante redactăm, în figura 2.3.13, algoritmul.

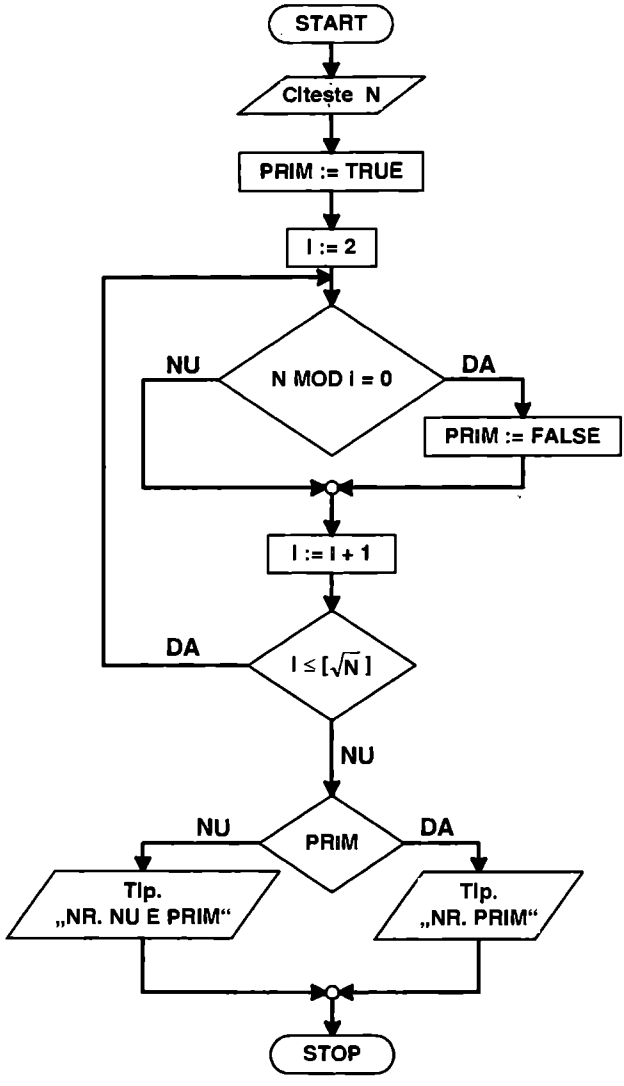


Figura 2.3.13.

Același algoritm îl redactăm în pseudocod:

```
INTEGER n,i
BOOLEAN prim;
READ n
prim:=true
FOR i:=2,√n
  IF n MOD i=0 THEN prim:=FALSE
  ENDIF
```

```
REPEAT
IF prim  THEN WRITE 'numarul este prim'
         ELSE WRITE 'numarul nu este prim'
ENDIF
STOP.
```

Algoritmul prezentat are un neajuns. Să presupunem că dorim să verificăm dacă numărul 1000 este prim. Primul divizor posibil (2) divide acest număr. Cu toate acestea, în loc să se afișeze rezultatul imediat, se verifică și ceilalți divizori posibili. Trebuie găsită o modalitate prin care, la găsirea unui divizor, să se tipărească faptul că numărul nu este prim.

Prezentăm acest nou algoritm în figura 2.3.14.

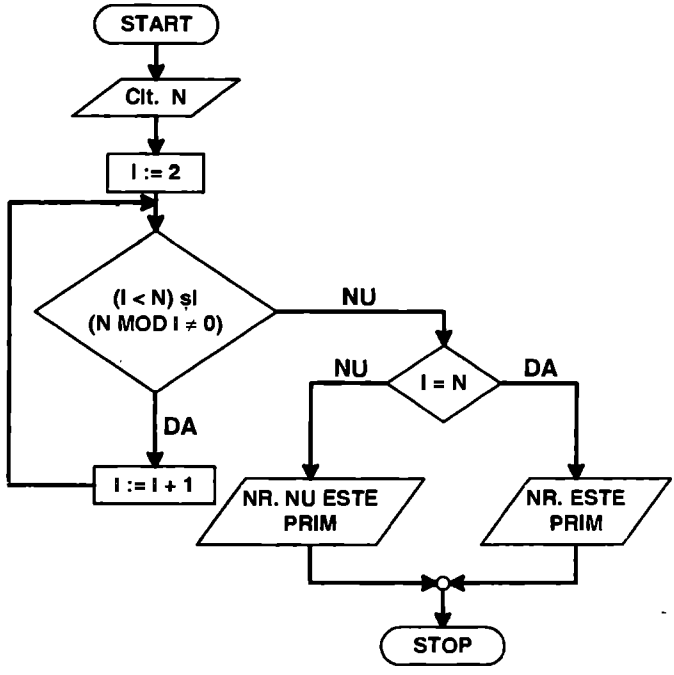


Figura 2.3.14.

Iată algoritmul redactat în pseudocod:

```
INTEGER n,i;
READ n
i:=2
WHILE (i<n) AND (n MOD i<>0)
  i:=i+1
REPEAT
IF i=n   THEN WRITE 'numarul este prim'
         ELSE WRITE 'numarul nu este prim'
ENDIF
STOP.
```

Acest algoritm are dezavantajul că, atunci când numărul este prim, se

încearcă toți divizorii posibili cuprinși între 2 și n–1. Un alt dezavantaj este acela că pentru n = 1 se tipărește mesajul corespunzător pentru număr neprim. Puteți găsi un algoritm care să elimine aceste dezavantaje?

Problema 9

Să se scrie un algoritm care tipărește primele nr numere prime (nr se citește).

În problema anterioară, am văzut cum putem stabili dacă un număr este prim. Acum, se pune problema să asigurăm un contor în care să se caute numerele prime printre numerele 1, 2, ș.a.m.d., până se tipăresc cele nr cerute (figura 2.3.15.).

În pseudocod, algoritmul arată astfel:

```
INTEGER n,i,j,nr
BOOLEAN prim;
READ nr
n:=1;
j:=0;
DO
  prim:=true
  FOR i:=2,[√n]
    IF n MOD i =0 THEN prim:=false
    ENDIF
  REPEAT
  IF prim THEN
    WRITE n
    j:=j+1
  ENDIF
  n:=n+1
UNTIL j=nr
STOP.
```

Problema 10

Se citește n, număr natural. Să se descompună în factori primi.

Algoritmul constă în următoarele:

- se citește n;
- se inițializează primul divizor (i = 2);
- secvența următoare se repetă până când n = 1:
  - se inițializează fm cu 0 (factor de multiplicitate, arată puterea la care se găsește factorul prim);
  - atât timp cât i îl divide pe n, se execută următoarele:
    - se mărește cu 1 factorul de multiplicare;
    - se împarte n la i;
  - dacă fm este diferit de 0 (deci i a fost găsit divizor al lui n) se tipăresc i și fm;
  - se adună 1 la i (se incrementează).

Să presupunem că am citit n = 12. Algoritmul decurge astfel:

- se inițializează i cu 2;
- se inițializează fm cu 0;
- întrucât 2 divide pe 12, fm va lua valoarea 1 iar n valoarea 6;
- întrucât 2 divide pe 6, fm va lua valoarea 2 și n valoarea 3;

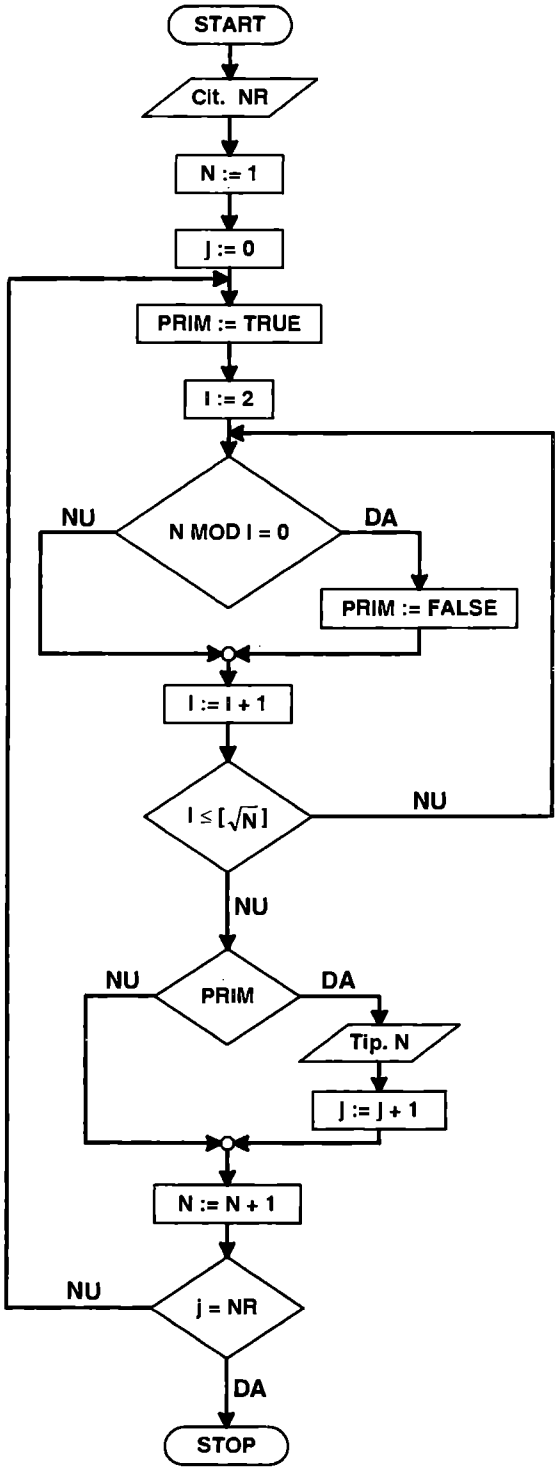

START
Cit. NR
N := 1
J := 0
PRIM := TRUE
I := 2
NU
N MOD I = 0
DA
PRIM := FALSE
I := I + 1
I ≤ [√N]
NU
NU
NU
PRIM
DA
Tip. N
J := J + 1
N := N + 1
NU
j = NR
DA
STOP

Figura 2.3.15.

- 2 nu divide pe trei, deci se trece mai departe;
- fm este diferit de 0, se tipărește 2 'la puterea' 2;
- se mărește i cu 1;
- n nu este 1, deci secvența se reia;
- fm va lua valoarea 0;
- 3 divide pe 3 (i divide pe n) deci fm va lua valoarea 1 și n va deveni 1;
- 3 nu divide pe 1, deci se trece mai departe;
- fm este diferit de 0, deci se tipărește 3 'la puterea' 1;
- i devine 4;
- n este 1 și algoritmul se încheie.

În continuare se prezintă schema logică și algoritmul realizat în pseudocod.

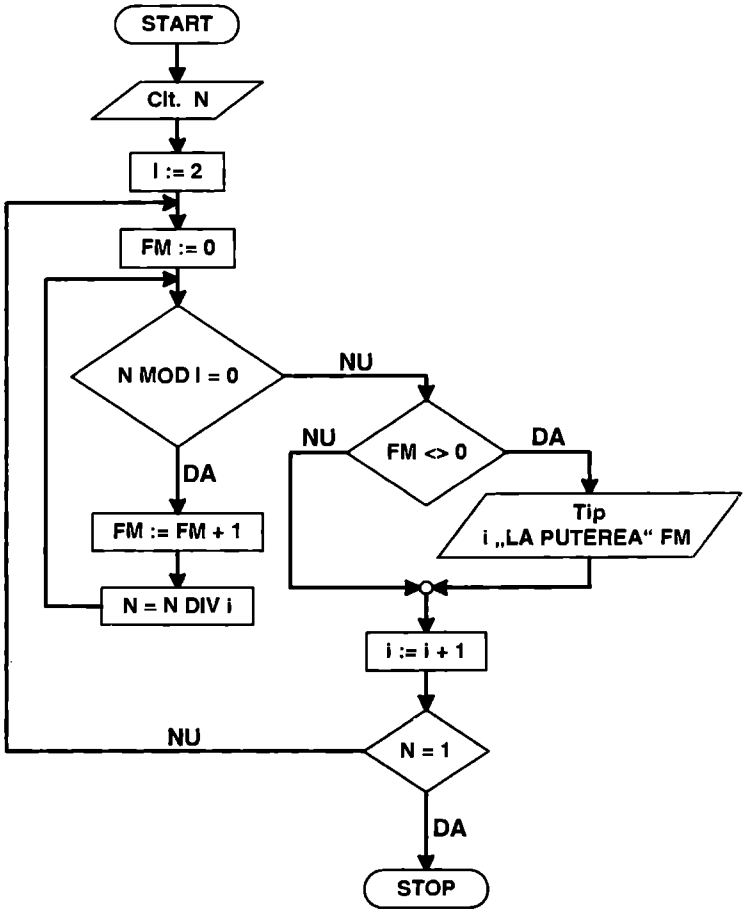


Figura 2.3.16.

```
INTEGER n,i,fm;
READ n
i:=2;
DO
```

```
    fm:=0;
    WHILE n MOD i =0
      fm:=fm+1;
      n:=n div i
    REPEAT
    IF fm<>0 THEN
      WRITE i, 'la puterea' fm
    ENDIF
    i:=i+1
  UNTIL n=1
  STOP.
```

Problema 11

Să se tipărească toate numerele naturale mai mici sau egale cu 10000 care se pot descompune în două moduri diferite ca sumă de cuburi.

Adevărata problemă constă în a afla dacă un număr natural se poate sau nu descompune în două feluri ca sumă de cuburi (de exemplu, $1729 = 1^3 + 12^3 = 9^3 + 10^3$ și este cel mai mic număr natural care admite o astfel de descompunere). Într-o structură repetitivă de tip FOR se încearcă, pe rând, să se descompună numerele între 1 și 10000.

Se consideră n, număr natural. Presupunând că acest număr se descompune în i + j, apare ca evident faptul că i și j nu pot lua valori mai mari decât radical indice 3 din n. Pe de altă parte, pentru a nu găsi o pereche (i,j) de două ori (de exemplu 3,5 și 5,3), i va lua valori între 1 și limita maximă, iar j va lua valori între i și limita maximă. O variabilă (nr în exemplul nostru) va reține numărul de perechi (i,j) găsite.

Prezentăm în continuare algoritmul în pseudocod și schema logică.

```
INTEGER n,nr,max,i,j,i1,j1,i2,j2;
FOR n:=1,10000
  max:=[∛n]
  nr:=0;
  FOR i:=1,max
    FOR j:=i,max
      IF i*i*i+j*j*j=n THEN
        IF nr=0 THEN
          i1:=i
          j1:=j
        ELSE
          i2:=i
          j2:=j
        ENDIF
        nr:=nr+1;
      ENDIF
    REPEAT
  REPEAT
  IF nr>1 THEN
    WRITE n,i1,j1,i2,j2
  ENDIF
REPEAT
STOP.
```

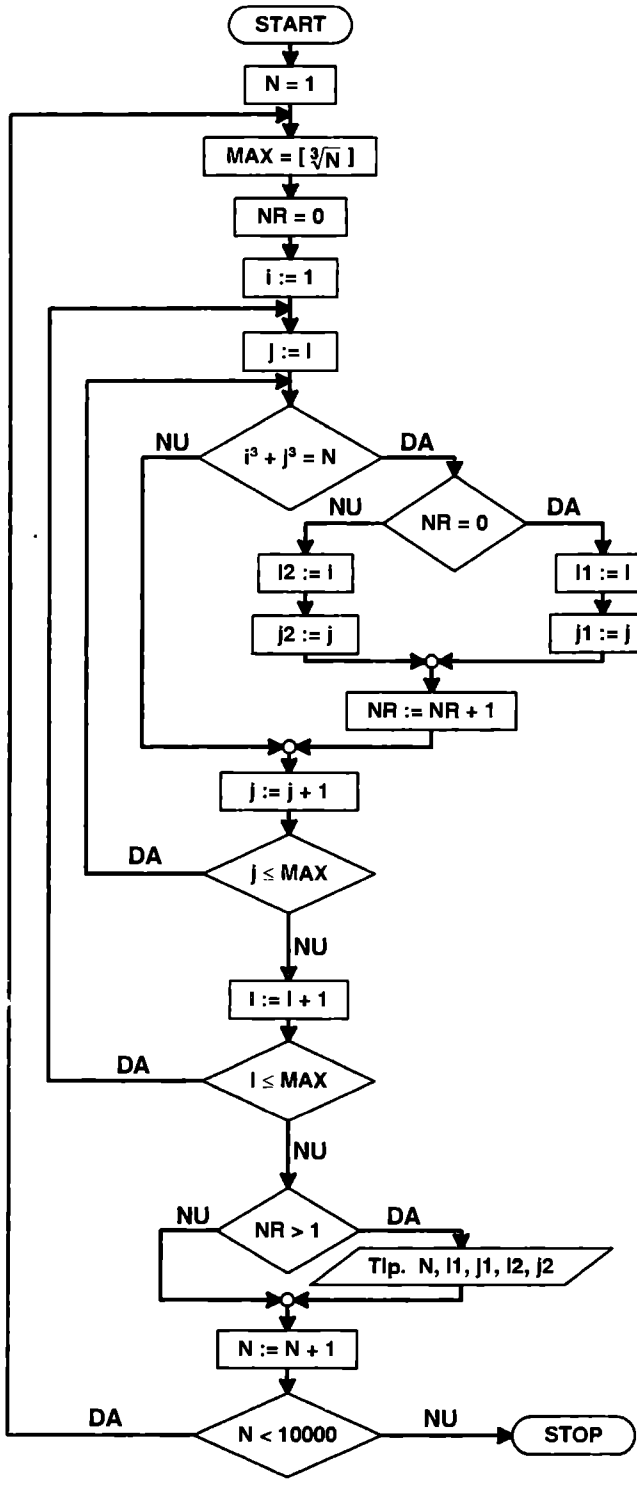

START
N = 1
MAX = [∛N]
NR = 0
i := 1
j := I
NU
i³ + j³ = N
DA
NU
NR = 0
DA
I2 := I
j2 := j
I1 := I
j1 := j
NR := NR + 1
j := j + 1
DA
j ≤ MAX
NU
I := I + 1
DA
I ≤ MAX
NU
NU
NR > 1
DA
Tip. N, I1, j1, I2, j2
N := N + 1
DA
N < 10000
NU
STOP

Figura 2.3.17.

Capitolul 3

Elemente de bază ale limbajului Pascal

3.1. Noțiuni introductive

3.1.1. Evoluția limbajelor de programare

Evoluția limbajelor de programare cunoaște trei perioade distincte de timp și anume: anii 1950-1959, 1960-1969, după 1970.

Anii '50 reprezintă etapa de pionierat în domeniul programării calculatoarelor. În această perioadă s-a cristalizat noțiunea de **program**. Inițial, programele erau scrise în **cod mașină** (instrucțiunile, datele de intrare-ieșire erau formate din succesiuni de 0 și 1, așa cum de fapt le prelucrează orice calculator din lume). Datorită acestui fapt, scrierea programelor, introducerea datelor erau activități deosebit de migăloase și puțin productive. Un pas important în evoluție este dat de apariția **limbajelor de asamblare**. Acestea folosesc pentru fiecare instrucțiune în cod mașină o abreviere din limba engleză ce sintetizează efectul instrucțiunii respective. Deși mult ușurată, activitatea de programare într-un limbaj de asamblare rămâne dificilă. Cu toate acestea, și astăzi limbajele de asamblare sunt utilizate cu succes datorită unor avantaje cum ar fi: posibilitatea scrierii unor programe specifice calculatorului pe care se rulează un anumit program (deci cu mare viteză de execuție), posibilitatea scrierii unor secvențe de program care nu se pot scrie într-un limbaj evoluat (din lipsa unor instrucțiuni specifice) etc.

În această perioadă apar și **limbajele de nivel înalt** (FORTRAN, COBOL). De această dată, scrierea programelor se face cu ajutorul unor expresii din limba engleză (deși extrem de restrictive). De asemenea, formulele prin care se solicită calculul unor expresii sunt asemănătoare formulelor matematice. Un mare avantaj al limbajelor de nivel înalt îl constituie **portabilitatea** programelor scrise cu ajutorul lor (posibilitatea ca un program scris să poată fi rulat pe diverse tipuri de calculatoare).

În anii '60 se dezvoltă cu precădere aparatul matematic necesar creării și utilizării limbajelor de programare. Astfel, regulile de scriere a programelor în limbajele respective sunt formalizate matematic. De asemenea, apare **conceptul de recursivitate** în programare (acest concept era utilizat de mult timp în matematică). Pentru prima dată apare noțiunea de **alocare dinamică** a memoriei (alocarea spațiului de memorie pentru diverse variabile se face în timpul execuției programului, atunci când este cazul, și tot în acest timp se poate șterge un spațiu alocat, când nu mai este necesar). Toate acestea apar pentru prima dată la limbajul ALGOL. Acest limbaj, mai puțin utilizat de practicanți, a avut un rol fundamental în dezvoltarea ulterioară a informaticii. Tot în această perioadă au apărut și alte limbaje de programare evoluate care s-au răspândit în întreaga lume.

La începutul anilor '70 a apărut **programarea structurată** (inițiatorii ei fiind E. W. Dijkstra și C. A. Hoare). Pentru prima dată apar metode standardizate de elaborare a programelor, activitatea de programare depășind stadiul artizanal. Redactarea oricărui program se poate face utilizând câteva structuri simple (secvențială, alternativă, repetitivă). O problemă complexă se descompune în subprobleme, care la rândul lor se descompun din nou, până când sunt suficient de simple pentru a fi transpuse în programe (programare descendentă).

La sfârșitul anilor '80 apare un nou concept și anume **programarea pe obiecte** (în care datele și rutinele pentru ele sunt unificate în 'obiecte' care se pot utiliza ca atare sau dezvolta ulterior fără a fi nevoie de a lua munca de la capăt sau de a le cunoaște în mecanismul lor intim). Timpul va decide dacă acestea vor deveni sau nu indispensabile în activitatea de programare.

Limbajul Pascal a apărut la începutul anilor '70 și a fost elaborat de matematicianul N. Wirth. Inițial, limbajul a fost conceput pentru predarea sistematică a disciplinei de programare a calculatoarelor (structurile clasice din programarea structurată au fost transformate în instrucțiuni). Cu timpul, limbajul a început să fie folosit și în activitatea practică de programare a calculatoarelor. Un rol fundamental pentru aceasta l-a avut firma americană Borland, care a implementat o variantă numită **Turbo Pascal**, care pe lângă instrucțiunile clasice ale limbajului conține și multe altele. Un mare avantaj al acestui limbaj este acela că utilizatorul are posibilitatea să-și declare propriile tipuri de date. Ultimele versiuni ale limbajului permit și realizarea programării pe obiecte.

3.1.2. Structura programelor Pascal

Să considerăm cel mai simplu program Pascal:

```
program fara_actiune;
begin
end.
```

Acesta este un program care se poate rula. Efectul său este nul. Nu citește, nu scrie, nu efectuează nici un calcul. Totuși, din analiza acestuia rezultă următoarele:

- orice program începe printr-un cuvânt numit **PROGRAM** care este urmat de numele propriu-zis al programului (nume ales de utilizator) și de semnul ';' ;
- orice program conține cel puțin o dată cuvintele cu un înțeles special **BEGIN** și **END**;
- orice program se termină prin punct.

Observații:

- orice cuvânt al programului poate fi scris cu litere mari sau mici (nu are importanță);
- în versiunea Turbo, prima linie a programului poate lipsi (deși nu este recomandabil acest lucru, din rațiuni de ordine);
- plasarea cuvintelor pe linie și numărul de spații dintre ele sunt la alegerea programatorului (putem scrie întreg programul pe o singură linie, însă este bine ca programul să fie scris în așa fel încât să fie ușor de înțeles).

Un program scris în Pascal (oricât de complex ar fi) are structura următoare:

PROGRAM nume;
- definiții de constante;
- definiții de tipuri;
- declarații de variabile;
- declarații de proceduri și funcții;

BEGIN
- instrucțiuni

END.

Nu este obligatoriu ca într-un program să figureze toate acestea, dar dacă ele figurează, trebuie să apară în această ordine.

3.1.3. Descrierea sintaxei cu ajutorul diagramelor de sintaxă

Orice limbaj de programare se caracterizează prin sintaxă și semantică. **Sintaxa** limbajului este dată de totalitatea regulilor de scriere corectă (în sensul acceptării sale de programul traducător (compilator) care are rolul de a converti programul în cod mașină pentru a fi executat). Dar un program corect din punct de vedere sintactic nu este automat un program bun. Corectitudinea sintactică este numai o cerință a programelor, tot așa cum pentru a fi campion mondial la alergări este necesar să nu ai picioarele amputate. Cel mai greu este ca programul să execute întocmai ce și-a propus cel care l-a realizat. Prin **semantica** unui limbaj se înțelege ce anume realizează fiecare instrucțiune a sa.

Sintaxa este formalizată perfect din punct de vedere matematic dar nu același lucru se întâmplă cu semantica. Sintaxa poate fi descrisă cu ajutorul **diagramelor de sintaxă**. Acestea nu reprezintă singura formă de descriere a sintaxei unui limbaj (mai există de exemplu **forma BNF**), dar se remarcă prin faptul că sunt sugestive.

În ce constă o descriere cu ajutorul diagramelor de sintaxă?

Să presupunem că dorim să descriem riguros cum arată scris un număr în baza 16. Dacă folosim o descriere de genul: un număr în baza 16 are în componența sa cifre și litere, cifrele pot fi de la 0 la 9 iar literele a, b, c, d, e, f, A, B, C, D, E, F și dăm câteva exemple, nu suntem suficient de riguroși. Imediat pot apărea întrebări de genul: A reprezintă un număr în baza 16? Mai mult, dacă dorim să facem un program care să recunoască dacă un număr citit este sau nu în baza 16, situația se complică. Reluăm descrierea unui număr în baza 16, cu ajutorul diagramelor de sintaxă.

Definim o cifră după cum se vede mai jos:

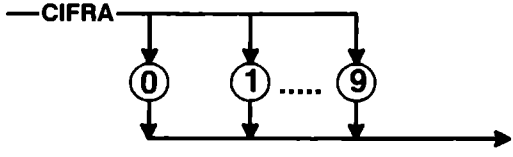


Figura 3.1.3.1.

Se observă că o astfel de schemă are un nume (în cazul nostru cifră). Pentru a putea obține o cifră trebuie să urmez un drum prin acest desen (în matematică se numește graf orientat), de la intrare (din dreptul numelui) până la ieșire, urmând sensul săgeților. Problema se poate pune și invers: fiind dat un caracter oarecare să se precizeze dacă este sau nu cifră. Caracterul dat este cifră dacă în diagrama prezentată există un drum care trece printr-un cerculeț ce prezintă caracterul.

Acum am definit riguros cifra. Putem defini riguros cifra hexa:

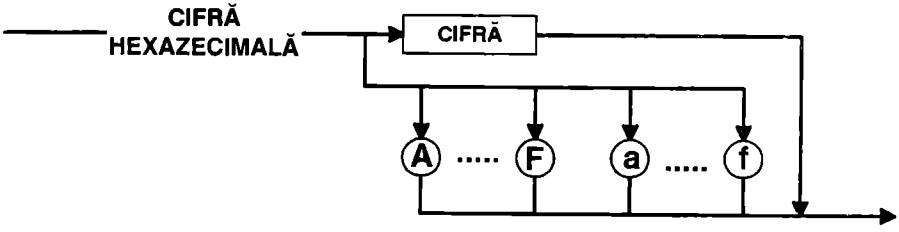


Figura 3.1.3.2.

În noua diagramă apare un dreptunghi în care este scris cuvântul CIFRA. Dreptunghiul se folosește atunci când se folosește o noțiune ce a fost definită printr-o altă diagramă de sintaxă. Orice drum folosim în acest desen (de la intrare la ieșire) obținem sau o cifră sau una din literele care pot face parte din structura unui număr în hexa.

Acum se poate defini un număr hexa:

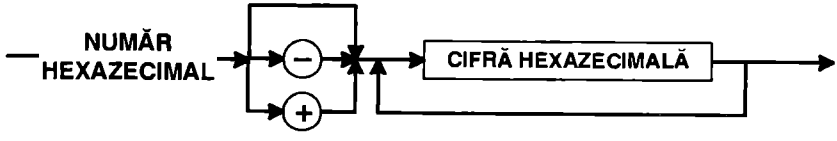


Figura 3.1.3.3.

În acest ultim desen observăm că este permisă de mai multe ori trecerea prin același loc.

În esență, într-o diagramă de sintaxă putem întâlni următoarele simboluri grafice:

- cercuri - încadrează anumite simboluri speciale;
- elipse - încadrează cuvinte rezervate (nu pot fi folosite în alt context);
- dreptunghiuri - încadrează elemente definite prin alte diagrame de sintaxă.

Ne-am putea întreba care este motivul unei descrieri atât de riguroase? Un program scris într-un limbaj de programare trebuie tradus de către programul numit **compilator** în cod mașină. În primul rând programul trebuie să fie corect din punct de vedere sintactic (altfel nu poate fi tradus). A verifica corectitudinea nu este un lucru simplu (pot fi foarte multe forme de programe). Din acest

motiv, verificarea se face după reguli care trebuie descrise foarte clar iar această descriere este realizată de diagramele de sintaxă.

3.2. Vocabularul limbajului

Vocabularul oricărui limbaj este format din:

- setul de caractere;
- identificatori;
- separatori;
- comentarii;

3.2.1. Setul de caractere

Reprezintă ansamblul caracterelor cu ajutorul cărora se poate realiza un program Pascal.

Acesta este alcătuit din:

- literele mari și mici ale alfabetului englez (A-Z, a-z);
- cifrele sistemului de numerație în baza 10 (0-9);
- caractere speciale: +, −, *, /, =, ^, <, >, (,), [,], {, }, ., ,, :, ;, #, $, @, _, și blanc (spațiu).

3.2.2. Identificatori

Prin identificator înțelegem o succesiune de litere sau cifre sau caracterul special '_' din care prima trebuie să fie în mod obligatoriu literă. Cu ajutorul identificatorilor se asociază nume constantelor, variabilelor, procedurilor etc.

Exemple de identificatori : a1, tasta, un_număr.

Contraexemple: 1ar, mt& (primul începe cu o literă, al doilea conține un caracter special).

Diagrama de sintaxă a unui identificator este următoarea:

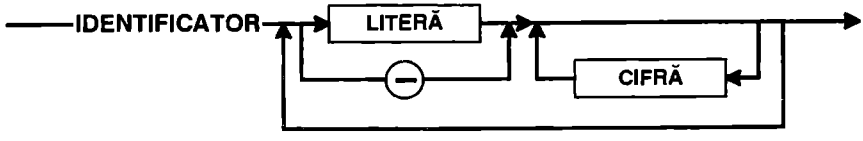


Figura 3.2.2.1.

O categorie specială de identificatori este dată de **cuvintele cheie** ale limbajului (au un înțeles bine definit și nu pot fi folosite în alt context). Acestea sunt: and, array, begin, case, const, div, do, downto, else, end, file, for, function, goto, if, in, label, mod, nill, not, procedure, program, record, repeat, set, of, or, origin, otherWise, packed, then, to, type, until, var, while, with.

3.2.3. Separatori și comentarii

Cele mai simple elemente alcătuite din caractere cu semnificație lingvistică poartă denumirea de unități lexicale. Acestea se separă între ele, după caz, prin blanc, sfârșit de linie sau caracterul ';'.

Pentru ca un program să fie ușor de înțeles se folosesc comentariile. Acestea se plasează oriunde în program. Un comentariu poate fi scris în două feluri:

- între acolade, exemplu: {un comentariu};
- între paranteze rotunde urmate de *, exemplu: (* un comentariu *).

3.3. Constante

Constantele Pascal reprezintă valori care pot fi conținute de un program scris în acest limbaj (nu sunt citite de program). Ele folosesc în calculul diverselor expresii (numerice, logice, șiruri de caractere) sau pentru scrierea unor mesaje.

Exemplu: În program trebuie realizată atribuirea y:=2*x+1. În acest caz, 2 și 1 sunt constante.

Constantele limbajului Pascal se împart în 4 categorii:

- constante întregi;
- constante reale;
- constante șir de caractere;
- constante simbolice.

Prezentarea noțiunii de constantă prin utilizarea unei diagrame de sintaxă arată astfel:

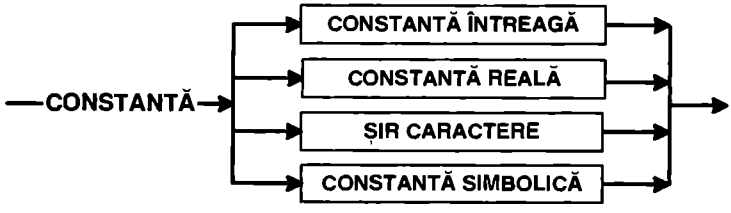


Figura 3.3.1.

3.3.1. Constante întregi

Sunt alcătuite dintr-o submulțime a numerelor întregi care pot fi reprezentate în memoria calculatorului.

Observație: Nu se pot reprezenta în calculator numere oricât de mari sau oricât de mici (orice număr ocupă un spațiu în memoria internă a calculatorului). Prin utilizarea limbajului Turbo Pascal se pot reprezenta numere întregi cuprinse în intervalul [–2.147.483.648, 2.147.683.647].

Numerele se pot reprezenta în baza 10 sau baza 16 (mai rar utilizată). Pentru baza 16, numărul este precedat de caracterul special '$'.

Exemple: 32, −164, +31, $a1, −a1.

Pentru eliminarea oricăror ambiguități, utilizăm diagrama de sintaxă pentru a defini un astfel de număr.

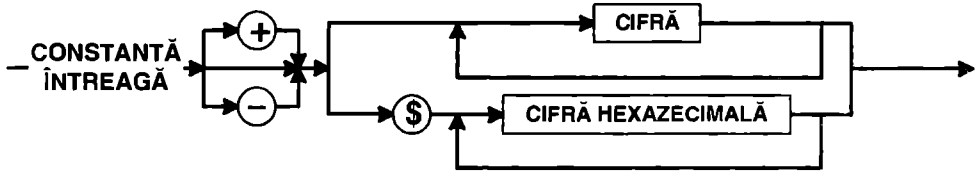


Figura 3.3.1.1.

3.3.2. Constante reale

Sunt alcătuite dintr-o submulțime a numerelor reale (mai precis a numerelor raționale) care pot fi reprezentate în calculator. Numerele se găsesc în intervalul $[3{,}4 \cdot 10^{-4352}, 1{,}1 \cdot 10^{4932}]$.

În locul virgulei se folosește punctul.

Exemple: 2.34, −45.26, 512E+23, −45.1E−3.

Ultimele două numere folosesc o scriere neîntâlnită în matematică. Ele reprezintă numerele $521 \cdot 10^{23}$ și $-45{,}1 \cdot 10^{-3}$.

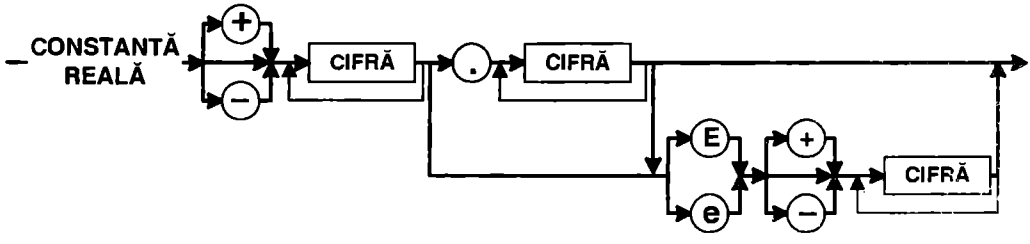


Figura 3.3.2.1.

3.3.3. Constante șir de caractere

După cum reiese și din denumire, cu ajutorul lor se reprezintă șiruri de caractere. Șirurile de caractere se pot reprezenta în două forme.

În prima formă se scrie șirul cuprins între apostrofuri.

Exemplu: 'acesta este un text'.

În forma a doua se scrie șirul în **cod ASCII** (fiecărui caracter i se atașează un cod cuprins între 0 și 255 și este precedat de caracterul '#'.

Exemplu: #1#2 sau, echivalent (în baza 16), #$1#$2.

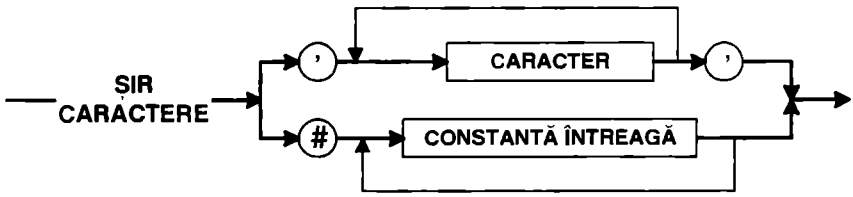


Figura 3.3.3.1.

3.3.4. Constante simbolice

Acestea sunt constante care au în program un anumit nume. Numele dat acestor constante poate fi cunoscut de limbaj (de exemplu: false, true, maxint) sau dat de utilizator atunci când definește constantele. Acest din urmă aspect îl vom analiza în paragraful 3.6.

3.4. Noțiunea de tip de dată

În procesul de calcul intervin diverse date (întregi, reale etc.) Prin **tip de dată** vom înțelege o mulțime de valori. Pe tipurile de date se introduc anumite operații.

Exemplu de tip: integer.

Mulțimea valorilor acestui tip este o submulțime a numerelor întregi. Cu aceste valori se pot efectua o serie de operații cum ar fi: adunarea, scăderea etc.

Avem trei categorii de tipuri:
- simple;
- structurate;
- reper.

Tipurile simple de date (nu necesită structurare) se pot clasifica din două puncte de vedere.

Un tip ordinal reprezintă o mulțime finită de valori între care există o relație de ordine.

Conform acestui criteriu, tipurile simple se clasifică în:
- tipuri ordinale;
- tipuri reale (chiar dacă se ia o submulțime a numerelor reale situată într-un anumit interval, acesta nu are un număr finit de elemente).

Din punct de vedere al modului de definire a tipului, tipurile simple se clasifică în:
- tipuri standard (este cunoscut, nu este necesar să fie definit de programator);
- tipuri definite de utilizator.

3.4.1. Tipuri simple standard

Acestea se clasifică în:

- tip boolean;
- tip char;
- tipuri întregi;
- tipuri real.

Dintre acestea, primele trei sunt tipuri ordinale.

3.4.1.1. Tipul boolean

Are numai două valori: **true** și **false**. Operatorii care lucrează cu acest tip sunt: **AND, OR, NOT**. În versiunea Turbo a limbajului s-a introdus și operatorul **XOR**. Fiind date două valori oarecare ale acestui tip, a și b, avem a **XOR** b = true dacă și numai dacă ele sunt diferite.

3.4.1.2. Tipul char

Mulțimea valorilor acestui tip este formată de toate caracterele reprezentate în codul ASCII extins. Fiecare din aceste caractere se reprezintă pe un octet. Relația de ordine între aceste caractere este dată de relația de ordine dintre codurile prin care se reprezintă aceste caractere (cuprinse între 0 și 255).

Limbajul conține anumite funcții care se pot aplica tipului char. Acestea sunt:

- SUCC(caracter) - returnează caracterul care are codul ASCII cu o unitate mai mare (succ('c') = 'd');
- PRED(caracter) - returnează caracterul care are codul ASCII cu o unitate mai mică (PRED('d') = 'c');
- ORD(caracter) - returnează codul ASCII al caracterului;
- CHR(cod) - returnează caracterul care are codul specificat.

3.4.1.3. Tipuri întregi

În funcție de submulțimea considerată a numerelor întregi, avem următoarea clasificare:

- tipul SHORTINT, numere din intervalul [–128, 127] care se reprezintă în memoria internă pe un octet;
- tipul INTEGER, numere în intervalul [–32768, 32767], se reprezintă pe 2 octeți;
- tipul LONGINT, numere din intervalul [–2.147.483.648, 2.147.483.647], se reprezintă pe 4 octeți;
- tipul BYTE, numere din intervalul [0, 255], se reprezintă pe un octet;
- tipul WORD, numere din intervalul [0, 65535], se reprezintă pe 2 octeți.

Variabilelor de tip întreg li se pot aplica următorii **operatori aritmetici**: **+** (adunare, operator binar), **–** (scădere, operator binar), * (înmulțire, operator binar), **DIV** și **MOD** (operatori binari). Acești ultimi doi operatori merită analizați.

În cazul a două valori naturale (deci întregi pozitive), operatorii MOD și DIV

furnizează câtul și restul împărțirii întregi.

Exemplu: 14 DIV 5 = 2;
14 MOD 5 = 4.

Dacă cel puțin un operand este negativ, lucrurile se complică.

Exemplu: −13 DIV 4 = −3
−13 MOD 4 = −1.

Știm din matematică faptul că restul împărțirii a două numere întregi este pozitiv. În acest caz, cei doi operanzi nu furnizează rezultatul corect.

Pentru operandul DIV rezultatul se obține astfel:

- se împart cele două numere în valoare absolută (pentru exemplul dat, câtul este 3);
- semnul câtului se stabilește după regula semnelor (+ cu + rezultat +, − cu + rezultat −, etc.).

Pentru MOD rezultatul se obține din scăderea din deîmpărțit a produsului dintre împărțitor și cât (pentru exemplificare se calculează −13−4*3 = −1.

Operanzilor de tip întreg li se pot aplica și **operatorii pe biți** (lucrează asupra biților din reprezentarea internă a numărului). Aceștia vor fi prezentați în continuare.

Operatorul NOT (operator unar)

Biții care au valoarea 0 vor avea valoarea 1 și invers.

Exemplu: NOT 7 = −8.

Presupunem numărul 7 reprezentat pe un octet. El va arăta astfel: 00000111. Aplicând operatorul NOT obținem: 11111000. Acesta reprezintă un număr negativ. După cum știm, pentru a obține valoarea lui, se transformă biții cu 1 în biți cu 0 și se adaugă 1. Rezultă: 00000111 + 1 = 00001000. Ultimul rezultat este numărul 8, iar dacă ținem seama că era negativ, obținem −8.

Operatorul AND (operator binar)

Fie conținutul a doi biți, b1 și b2. Definim operația de conjuncție asupra conținutului lor, astfel:

$$b1 \text{ AND } b2 = \begin{cases} 1, & \text{dacă } b1=1,\ b2=1 \\ 0, & \text{în orice alt caz} \end{cases}$$

Cu ajutorul lui AND se face conjuncția bit cu bit a celor doi operanzi.

Exemplu: 3 AND 7 = 3.

- 3 se reprezintă astfel: 00000011;
- 7 se reprezintă astfel: 00000111;
- în final, rezultă: 00000011, adică numărul 3.

Operatorul OR

Fie b1 și b2 conținutul a doi biți. Definim disjuncția astfel:

$$b1 \text{ OR } b2 = \begin{cases} 0, & \text{dacă } b1=0,\ b2=0 \\ 1, & \text{în orice alt caz} \end{cases}$$

Operatorul OR face disjuncția bit cu bit asupra reprezentării celor două numere.

Exemplu: 3 OR 7 = 7.

Operatorul XOR (operator binar)

Fie b1,b2 conținutul a doi biți. Definim operația *sau exclusiv* asupra conținutului celor doi biți astfel:

$$b1 \text{ XOR } b2 = \begin{cases} 1, & \text{dacă b1=1 şi b2=0 sau b1=0 şi b2=1} \\ 0, & \text{în caz contrar} \end{cases}$$

Prin aplicarea operatorului XOR se face operația *sau exclusiv* bit de bit asupra reprezentării binare a celor două numere.

Exemplu: 3 XOR 7 =4.

- reprezentăm numărul 3: 00000011;
- reprezentăm numărul 7: 00000111;
- rezultă: 00000100, adică 4.

Operatorul SHL (operator binar)

Deplasează la stânga toți biții primului operand, cu un număr de biți egal cu al doilea operand. Conținutul primilor biți se pierde, iar conținutul ultimilor biți devine 0.

Exemplu: 11 SHL 2 = 44.

- 11 se reprezintă 00001011;
- prin deplasarea cu două poziții la stânga se obține: 00101100, adică numărul 44.

Observație: Prin deplasarea la stânga cu o poziție (dacă primul bit al numărului nu este 1) numărul se înmulțește cu 2.

Operatorul SHR

Deplasează toți biții primului operand la dreapta cu un număr de poziții egal cu al doilea operand, ultimii biți ai primului operand se pierd, iar pe primele poziții se pune 0.

Exemplu : 44 SHR 2 = 11. Arătați cum s-a obținut acest rezultat.

Observație: Deplasarea la dreapta cu o poziție este echivalentă cu împărțirea întreagă la doi.

Tipurile întregi sunt tipuri ordinale deci li se pot aplica funcțiile SUCC, PRED, ORD și operatorii relaționali.

FUNCȚIA ABS(nr) - furnizează modulul numărului (abs(-12) = 12);

3.4.1.4. Tipuri reale

Într-o dată de tip real se pot reține numere raționale. Numerele reale, cunoscute de noi din matematică, cuprind și numere iraționale. Acestea din urmă au o infinitate de zecimale, deci nu pot fi reținute în calculator. Totuși, ele pot fi suficient de bine aproximate prin numere raționale. Operațiile cu numere reale au rezultate aproximative (nu se pot reține oricât de multe zecimale).
Limbajul TURBO PASCAL are următoarele tipuri reale:

- tipul REAL - numerele se reprezintă pe 6 octeți și pot avea maxim 11, 12 cifre;
- tipul SINGLE - numerele se reprezintă în memorie pe 4 octeți și pot avea maxim 7, 8 cifre;

- DOUBLE, numerele se reprezintă pe 8 octeți și pot avea maxim 15, 16 cifre;
- EXTENDED, numerele se reprezintă pe 10 octeți și pot avea 19, 20 de cifre;
- COMP se reține numai numere pozitive pe 8 octeți și numerele pot avea până în 19, 20 de cifre.

Datelor de tip real li se pot aplica următorii operatori:

\+ pentru adunare;
\- pentru scădere;
* pentru înmulțire;
/ pentru împărțire.

Datelor de tip real li se pot aplica operatorii relaționali și următoarele funcții:

- ABS(x) - are ca rezultat modulul numărului x;
- SQR(x) - x la pătrat;
- SQRT(x) - radical din x;
- SIN(x) - sinus din x (x este exprimat în radiani);
- COS(x) - cosinus din x;
- ARCTAN(x) - arctangent din x;
- LN(x) - logaritm natural din x;
- EXP(x) - e la puterea x (e are valoarea aproximativă 2.71);
- INT(x) - partea întreagă din x;
- FRAC(x) - partea fracționară a lui x;
- pi -numărul π (aproximativ 3.14).

O dată de tip real nu este de tip ordinal.

3.4.2. Tipuri ordinale definite de utilizator

Utilizatorul are posibilitatea să declare două tipuri de date ordinale:

- tipul enumerat;
- tipul subdomeniu.

3.4.2.1. Tipul enumerat

Având în vedere faptul că acest limbaj a fost creat în scopuri pur didactice, s-a avut în vedere posibilitatea scrierii unor programe care să fie înțeles cu ușurință. Din acest motiv a apărut tipul de date enumerat.

Exemple de astfel de tipuri:

- (luni, marți, miercuri, joi, vineri, sâmbătă, duminică);
- (ieri, azi, mâine).

Prin diagrame de sintaxă acest tip se reprezintă astfel:

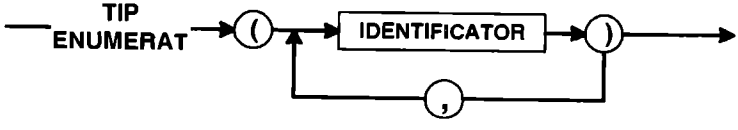


Figura 3.4.2.1.

Datele de acest tip nu pot fi scrise sau citite. Ele se reprezintă în memorie prin numărul lor de ordine (ORD(luni) = 0, ORD(marți) = 1 etc. Acestor date li se pot aplica funcțiile PRED, SUCC precum și operatorii relaționali (avem, referitor la exemplul dat, marți < miercuri)).

3.4.2.2. Tipul subdomeniu

Se obține dintr-un tip ordinal definit anterior sau predefinit (numit **tip de bază**) din care se extrage un număr de valori consecutive (precizând prima și ultima valoare). Cu acest tip se pot face aceleași operații care se pot face și cu tipul de bază).

Exemple:

- 2..20 - o variabilă de acest tip ia valori întregi cuprinse între 2 și 20, tipul de bază fiind tipul integer;
- luni..vineri - o variabilă de acest tip poate lua toate valorile dintre luni și vineri ale tipului de bază enumerat, definit anterior.

Prezentăm acest tip prin intermediul diagramelor de sintaxă:

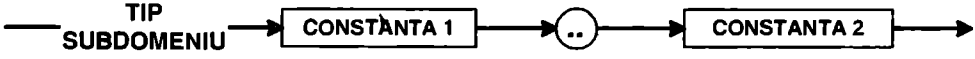


Figura 3.4.2.2.

3.4.3. Definirea tipurilor

După ce am văzut principalele tipuri de date ale limbajului, trebuie să cunoaștem și modul în care se fac declarațiile de tipuri.

Pentru declararea unui tip se folosește declarația TYPE.

Exemple: TYPE zile = (luni, marți, miercuri, joi, vineri, sâmbătă, duminică);
an_naștere = 1900..1994;
litere = 'A'..'Z';

Observații:

- într-un program putem defini mai multe tipuri de date prin utilizarea unui singur cuvânt TYPE (toate declarațiile de mai sus pot fi făcute într-un singur program);
- atunci când declarăm un tip, declarăm de fapt o mulțime de valori posibile pe care le-ar putea lua variabilele programului.

Declarația tipurilor cu ajutorul diagramelor de sintaxă o prezentăm mai jos.

3.5. Declararea variabilelor

Noțiunea de variabilă a fost prezentată în primul capitol. Aici se prezintă modul în care se declară o variabilă de un anumit tip. Pentru declarare se folosește cuvântul cheie **VAR**.

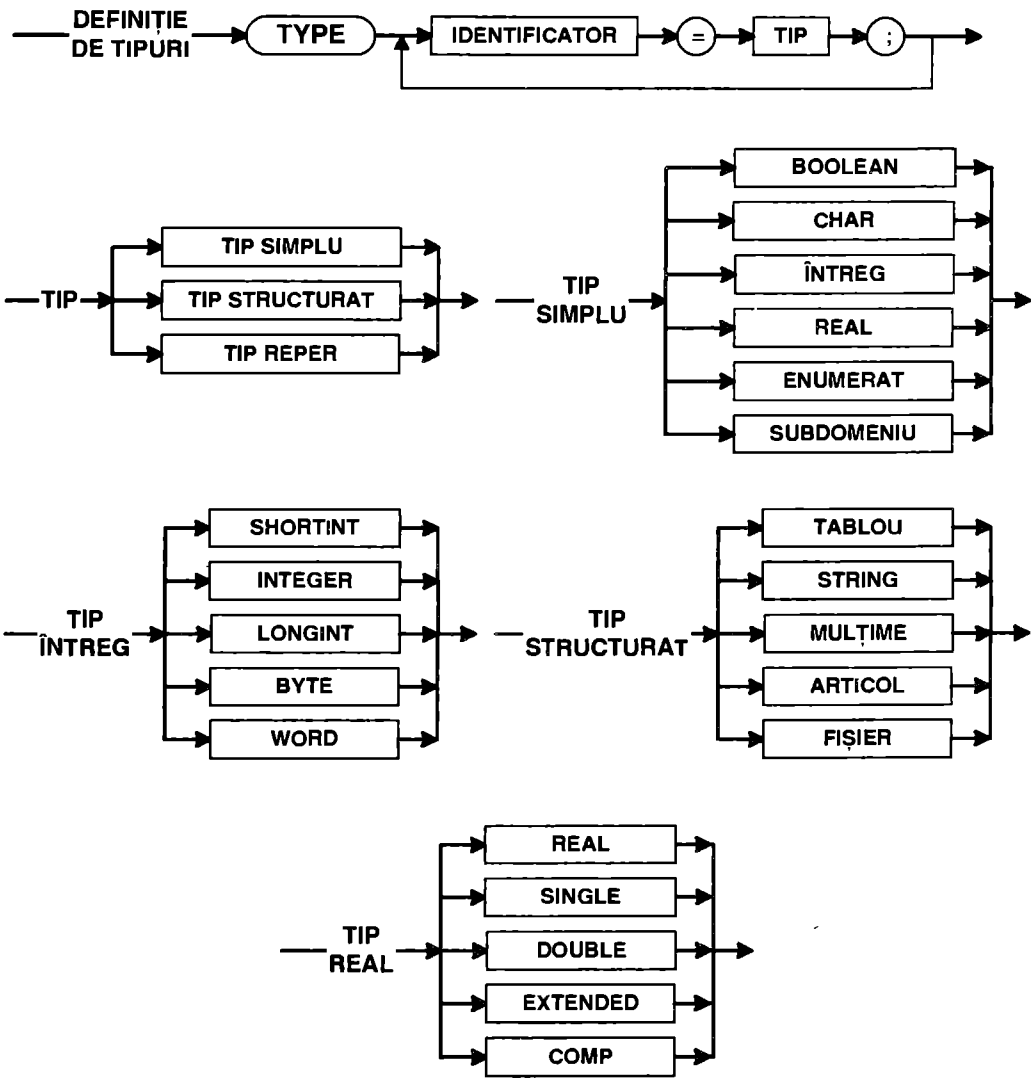


Figura 3.4.3.1.

Exemple: TYPE zile = (luni, marți, miercuri, joi, vineri, sâmbătă, duminică);
an_naștere = 1900..1994;
litere = 'A'..'Z';
Var a: integer;
zi: zile;
an1, an2: an_naștere;
ch: litere;

Prezentăm declararea variabilelor cu ajutorul diagramelor de sintaxă:

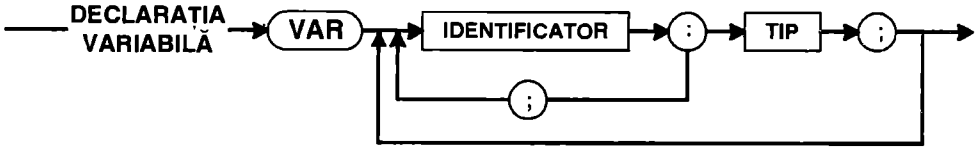


Figura 3.5.1.

3.6. Definirea constantelor

Se face prin utilizarea cuvântului cheie CONST.

Exemple: Const nota_max = 10;
r = 3.25.

Există posibilitatea de a defini o constantă prin precizarea tipului ei (care poate fi și un tip structurat). În acest manual se vor da astfel de exemple de câte ori va fi cazul.

3.7. Expresii

Regulile de formare a expresiilor sunt aceleași din matematică.

O mare importanță în evaluarea expresiilor o are **prioritatea operatorilor**. Aceasta este următoarea:

- prioritatea 1 (maximă): NOT, +, – (operatori unari);
- prioritatea 2: AND, *, /, DIV; (operatori multiplicativi);
- prioritatea 3: OR, XOR, +, – (operatori aditivi);
- prioritatea 4; > ,> =,<,< =,=, IN (operatori relaționali).

În mod normal, expresia se evaluează de la stânga la dreapta, ținând cont de prioritățile operatorilor (putem schimba prioritățile prin utilizarea parantezelor rotunde).

Greșeli posibile:

1. 1/2*a se calculează a de împărțit la 2 și nu 1 de împărțit la 2*a, dacă se dorește aceasta se pune 1/(2*a);
2. expresia (a>b) AND (b<c) nu este echivalentă cu expresia:
a>b AND b<c (aceasta din urmă nici nu are sens deoarece se efectuează inițial b AND b, după care expresia își pierde sensul).

3.8. Citirea și scrierea datelor

Acestea se realizează utilizând procedurile standard **READ, READLN, WRITE, WRITELN.**

3.8.1. Citirea datelor

Se face cu ajutorul procedurilor READ și READLN.

În acest paragraf se prezintă aceste proceduri într-o formă mult simplificată. Pentru amănunte, vezi fișiere text.

Procedura READ are forma READ(var), unde var are semnificația de variabilă (variabilele de tip boolean și enumerare nu se citesc). Se citește de la tastatură o dată care se depune în variabila *var*. Procedura READLN se folosește exact ca READ, cu deosebirea că, după citire, cursorul sare pe linia următoare a ecranului.

3.8.2. Scrierea datelor

Se face cu ajutorul procedurilor WRITE și WRITELN.

În ambele cazuri se scrie începând cu poziția curentă a cursorului pe ecran. Diferența între WRITE și WRITELN este aceea că, după scriere, prin utilizarea procedurii WRITE cursorul rămâne după ultimul caracter scris, iar prin utilizarea procedurii WRITELN el sare pe prima poziție a rândului următor.

Exemple:

- WRITE (a), unde a este o variabilă întreagă care conține numărul 3 - scrie 3 începând din poziția curentă a cursorului.
- WRITE (a,b), unde *a* și *b* sunt variabile întregi care au conținutul 2 respectiv 3 - va avea ca efect scrierea numărului 2 urmat de numărul 3 (apare 23).
- WRITE (' a = ',a) - are ca rezultat scrierea șirului de caractere 'a = ' urmat de data reținută de variabila întreagă *a*.
- Presupunem că *b* este o variabilă reală care conține numărul 3.25. WRITELN (b:4:2) va determina apariția pe monitor a numărului 3.25. Cifra 4 are semnificația de număr total de poziții pe care să se scrie numărul real, iar cifra 2 numărul de poziții pe care se va scrie partea zecimală (în absența acestor parametri, data reală se va scrie într-o formă greu inteligibilă). În situația în care partea întreagă a datei reale nu încape în formatul prestabilit, nu se mai ține cont de acest format.

Capitolul 4

Instrucțiunile limbajului Turbo Pascal

Înainte de a începe prezentarea instrucțiunilor acestui limbaj, amintim condițiile în care acesta a fost elaborat.

1. Limbajul a fost conceput special pentru scrierea de programe structurate. Din acest motiv, instrucțiunile reproduc structurile fundamentale ale programării structurate. Există și o excepție și anume instrucțiunea GO TO (salt la o anumită etichetă). Motivul includerii acesteia în setul de instrucțiuni ale limbajului este acela de a permite programatorilor, obișnuiți să lucreze nestructurat, folosirea acestui limbaj (o glumă care circula la apariția limbajului spune că această instrucțiune a fost introdusă spre a permite programatorilor să scrie programe BASIC în PASCAL). Întrucât elevilor nu le este permis să o folosească, nu va fi prezentată în acest manual.

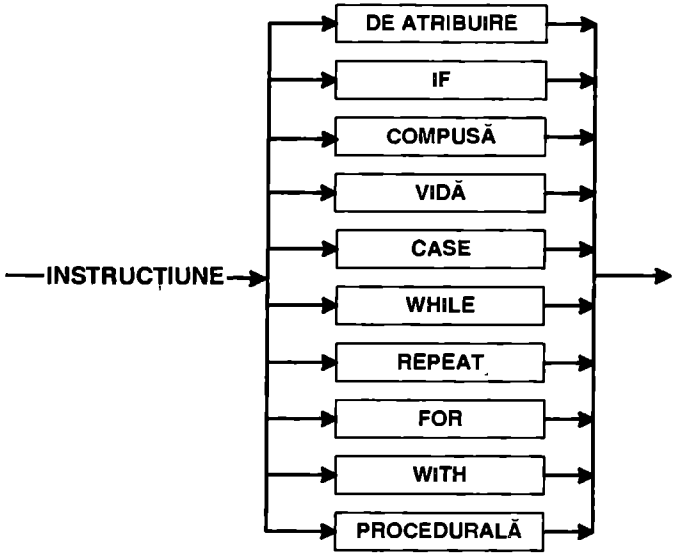


Figura 4.1.

2. Din fericire, autorul limbajului este de formație matematician, fapt ce i-a permis să realizeze următoarele:
- definirea riguroasă (matematică) a sintaxei limbajului;
- mari posibilități de extindere a tipurilor de date;
- introducerea instrucțiunii vide, cu mari consecințe asupra programelor.

3. Limbajul a fost conceput să fie unul didactic (introducerea în programare se face cu ajutorul lui, în multe universități).

Din acest motiv, el este extrem de apropiat de un limbaj de tip pseudocod, fapt care permite programatorilor cu anumită experiență să redacteze algoritmii direct în Pascal. Ulterior, datorită marilor sale calități, limbajul a fost extins la varianta Turbo Pascal, variantă ce este folosită cu succes de profesioniști.

Conceptul de instrucțiune rezultă din diagrama de sintaxă din figura 4.1.

4.1. Instrucțiunea de atribuire

Este de forma: v:=expresie, unde v este numele unei variabile.

Principiul de execuție este următorul:

- se evaluează expresia;
- variabila v ia această valoare.

Regula fundamentală este următoarea: tipul expresiei trebuie să coincidă cu tipul variabilei.

Exemple:

1. x:=3;
   x:=x+2 unde *x* este o variabilă de tip integer sau real.
   În urma execuției acestei secvențe, variabila x va conține valoarea 5.
2. a:=i<j unde a este o variabilă de tip boolean, i și j sunt variabile de tip real;
   Se evaluează expresia i<j . Dacă valoarea lui *i* este mai mică decât valoarea lui *j*, *a* va lua valoarea TRUE, în caz contrar *a* va lua valoarea FALSE.

Contraexemple:

1. i:=i+a unde *i* este o variabilă de tip integer, iar *a* de tip real. În acest caz expresia este de tip real iar variabila *i* este de tip integer. La compilare apare un mesaj de eroare ('type mismatch');
2. i:=i/2, unde *i* este de tip integer. Cu toate că rezultatul ar putea fi de tip integer (dacă *i* este par), forma nu este permisă pentru că expresia este de tipul real. Corect, ar fi trebuit scris i:= i div 2.

În diagrama de sintaxă prezentată în figura 4.2., apare și identificatorul de funcție. Astfel de atribuiri vor fi prezentate în capitolul care tratează subprogramele Pascal.

Programul p1 interschimbă conținutul a două variabile care au fost citite. La sfârșit, se tipărește noul conținut al variabilelor.

```
program p1;
var a,b,c:integer;

begin
  write('a=');
  readln(a);
```

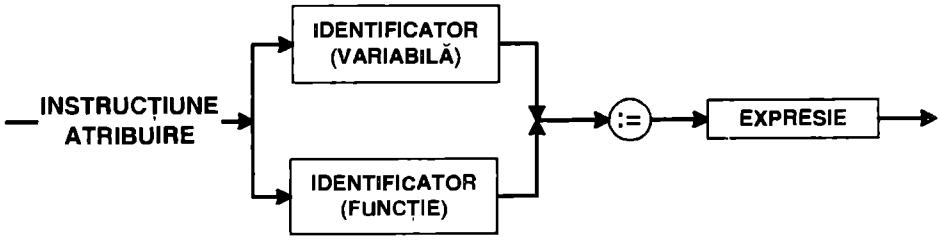


Figura 4.2.

```
  write('b=');
  readln(b);
  c:=a;
  a:=b;
  b:=c;
  writeln('a=',a);
  writeln('b=',b);
end.
```

4.2. Instrucțiunea IF

Această instrucțiune corespunde perfect structurii alternative.

Instrucțiunea IF are două forme: IF THEN ELSE și IF THEN.

4.2.1. IF THEN ELSE

Instrucțiunea se scrie astfel:

IF expresie logică THEN instrucțiune1
ELSE instrucțiune2

Principiul de execuție este următorul:

- se evaluează expresia logică;
- dacă aceasta ia valoarea TRUE, se execută instrucțiunea plasată după THEN, în caz contrar se execută instrucțiunea plasată după ELSE.

Atenție! Atât după THEN cât și după ELSE avem voie să punem o singură instrucțiune.

Exemplu: Programul care urmează calculează maximul dintre două numere citite:

```
program max;
var a,b:integer;

begin
  write('a=');
  readln(a);
  write('b=');
  readln(b);
  if a>b  then writeln(a)
```

```
            else writeln(b)
end.
```

Se citesc trei variabile reale a, b, și c. Se cere să se calculeze

$$e=\begin{cases} a+b, & c>0 \\ a*b, & c=0 \\ a-b, & c<0 \end{cases}$$

Programul p3, care rezolvă această problemă, exemplifică folosirea unei instrucțiuni IF, plasată în corpul altei instrucțiuni IF.

```
program p3;
var a,b,c,e:real;

begin
  write ('a=');
  readln(a);
  write ('b=');
  readln(b);
  write ('c=');
  readln(c);
  if c>0   then e:=a+b
           else if c=0  then e:=a*b
                        else e:=a-b;
  writeln('e=',e:3:2)
end.
```

4.2.2. Forma IF THEN

Această instrucțiune se scrie sub forma:

IF expresie logică THEN instrucțiune.

Principiul de execuție este următorul:

- se evaluează expresia logică;
- în situația în care aceasta ia valoarea TRUE, se execută instrucțiunea aflată după THEN, în caz contrar se trece la instrucțiunea următoare.

Iată și diagrama de sintaxă care prezintă instrucțiunea IF:

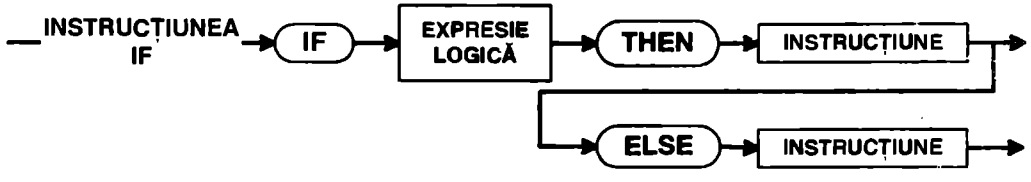


Figura 4.3.

4.3. Instrucțiunea compusă

Se pune următoarea problemă: cum procedăm în situația în care la utilizarea instrucțiunii IF după THEN urmează să scriem mai multe instrucțiuni (aceeași problemă apare dacă dorim să scriem mai multe instrucțiuni după ELSE). Pentru a putea scrie mai multe instrucțiuni care să fie interpretate de compilator ca una singură se folosește instrucțiunea compusă.

Instrucțiunea compusă are forma următoare:

```
begin
  i1;
  i2;
  in
  .
  .
end.
```

Aici, i1,..., in reprezintă instrucțiunile care se găsesc în corpul instrucțiunii compuse. Acestea sunt separate prin virgulă.

Exemplu: Să se scrie un program care rezolvă ecuația de gradul 1 ($a*x+b=0$). Modul de rezolvare a acestei ecuații a fost discutat pe larg atunci când am exemplificat elaborarea algoritmilor prin utilizarea schemelor logice și a limbajului de tip pseudocod. Aici prezentăm programul Pascal care rezolvă această problemă. Se observă folosirea unei instrucțiuni compuse la calculul rădăcinii și tipărirea ei.

```
program p2;
var a,b,x:real;

begin
  write('a=');
  readln(a);
  write('b=');
  readln(b);
  if a<>0 then
  begin
    x:=-b/a;
    writeln('x=',x:3:2);
  end
  else if b=0  then writeln('infinitate de solutii')
               else writeln('nu avem solutii')
end.
```

Observație: orice program PASCAL are cel puțin o instrucțiune compusă (începe cu BEGIN și se termină cu END).

Prezentăm diagrama de sintaxă a acestei instrucțiuni:

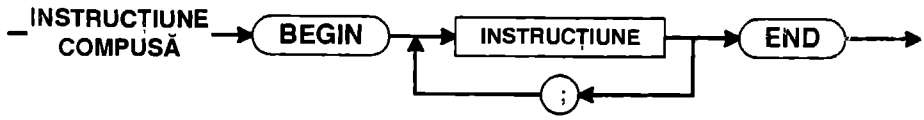


Figura 4.4.

4.4. Instrucțiunea vidă

Instrucțiunea vidă nu se trece în program. Totuși, prin faptul că este admisă ca instrucțiune, apar multe avantaje cum ar fi:

- în cadrul instrucțiunii compuse, între ultima instrucțiune și blocul END, sintaxa cere să nu se pună separatorul ';'. De multe ori, din obișnuință, programatorii pun chiar și în acest caz semnul ';'. Compilatorul nu semnalează acest fapt ca eroare, întrucât se presupune că între separator și END se găsește instrucțiunea vidă;
- se poate utiliza instrucțiunea IF sub forma:

```
IF expresie logică   THEN
                     ELSE instrucțiune;
```

(după THEN urmează de fapt instrucțiunea vidă).

4.5. Instrucțiunea CASE

Am arătat faptul că structurii alternative îi corespunde în limbaj instrucțiunea IF. Cum procedăm în situația în care în urma unui test avem mai multe variante de execuție? Desigur, putem folosi instrucțiunea IF (practic după THEN sau după ELSE se trec alte instrucțiuni IF). Această modalitate de lucru se dovedește de multe ori ineficientă. Acesta este motivul includerii în setul de instrucțiuni al limbajului a instrucțiunii CASE.

Forma generală a acesteia este:

```
CASE expresie ordinală do
   c1,[c2,...,cn1] : instrucțiune 1;
   p1,[p2,...,pn2] : instrucțiune 2;
   z1,[z2,...,znp] : instrucțiune p
   [ ELSE instrucțiune]
end.
```

Aici, c1,...,cn1,...,znp reprezintă constante de același tip ca și expresia ordinală.

Principiul de execuție este următorul:

- se evaluează expresia ordinală;
- se execută acea instrucțiune care are în față constanta obținută în evaluarea expresiei;

- în situația în care nici una din instrucțiunile 1...p nu este precedată de acea constantă, se execută instrucțiunea plasată după ELSE;
- dacă și clauza ELSE este absentă, se trece la instrucțiunea următoare.

Observații:
- constantele trebuie să fie distincte;
- pot exista cel mult 255 constante;

Exemplu: Programul care urmează probează instrucțiunea CASE.

```
program c1;
var i:integer;

begin
  write('i=');
  readln(i);
  case i of
    1:     writeln('am citit 1');
    2,3:   writeln('am citit 2 sau 3');
    else   writeln('am citit altceva')
  end;
end.
```

Formal, instrucțiunea CASE se prezintă astfel:

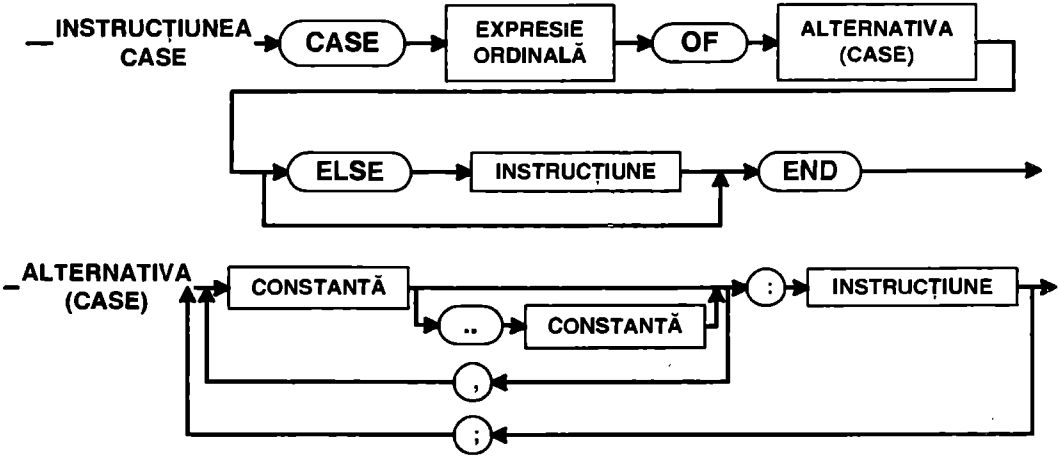


Figura 4.5.

4.6. Instrucțiunea WHILE

Această instrucțiune reproduce structura de tip WHILE DO.

Forma generală este:

WHILE expresie logică DO instrucțiune.

Principiul de execuție este următorul:

- se evaluează expresia logică și în caz că aceasta ia valoarea TRUE se execută instrucțiunea, se evaluează din nou expresia, dacă aceasta ia valoarea TRUE, se execută din nou instrucțiunea; procedeul se repetă până când la evaluarea expresiei se obține FALSE.

În situația în care se dorește să se ruleze mai multe instrucțiuni când expresia a luat valoarea TRUE, se folosește instrucțiunea compusă.

Pentru următoarele exemple, algoritmii de rezolvare au fost prezentați în capitolul 2.

Exemplul 1:

Se citește n, număr natural. Să se calculeze suma cifrelor sale (pentru n = 213, se va tipări 6).

```
program p6;
var n,s:integer;

begin
  write('n=');
  readln(n);
  s:=0;
  while n<>0 do
  begin
    s:=s+n mod 10;
    n:=n div 10
  end;
  writeln('s=',s)
end.
```

Exemplul 2:

Se citește n, număr natural. Să se tipărească numărul obținut prin inversarea cifrelor sale (pentru n = 412, se va tipări 214).

```
program p7;
var n,ninv:integer;

begin
  write ('n=');
  readln(n);
  ninv:=0;
  while n<>0 do
  begin
    ninv:=ninv*10+n mod 10;
    n:=n div 10
  end;
  writeln ('numarul inversat =', ninv);
end.
```

Instrucțiunea WHILE se prezintă prin diagrama de sintaxă astfel:

Figura 4.6.

4.7. Instrucțiunea REPEAT

Această instrucțiune reproduce structura REPEAT UNTIL. Forma generală a acestei instrucțiuni este următoarea:

```
repeat
  i1;
  i2;
  i3;
  .
  .
  in
until expresia logică.
```

Aici, i1,...,in reprezintă instrucțiuni.

Principiul de execuție este următorul:

- se efectuează secvența de instrucțiuni i1,...,in până când, la evaluare, expresia logică ia valoarea TRUE.

Observație: Secvența se execută cel puțin o dată, după care se pune problema dacă să se repete sau nu (prin evaluarea expresiei logice).

Exemplul 1:

Se citește un număr natural n, mai mare sau egal cu 1. Să se calculeze suma primelor n numere naturale.

```
program suma;
var n,s,i:integer;

begin
  write('n=');
  readln(n);
  i:=1;
  s:=0;
  repeat
    s:=s+i;
    i:=i+1
  until i>n;
  writeln('s=',s)
end.
```

Exemplul 2:

Se citește n, număr natural. Să se descompună în factori primi.

```
program p10;
var n,i,fm:integer;

begin
  write('n=');
  readln(n);
  i:=2;
  repeat
    fm:=0;
    while n mod i =0 do
    begin
      fm:=fm+1;
      n:=n div i
    end;
    if fm <>0 then writeln (i, ' la puterea ',fm);
    i:=i+1
  until n=1
end.
```

Iată cum arată instrucțiunea REPEAT prezentată cu ajutorul diagramei de sintaxă:

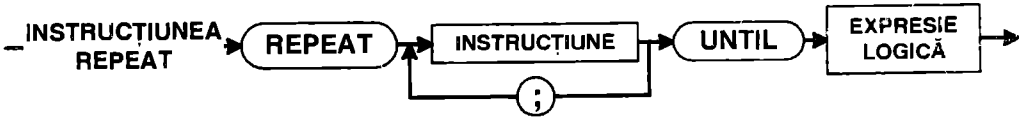


Figura 4.7.

4.8. Instrucțiunea FOR

De multe ori trebuie repetată o anumită secvență de un număr cunoscut de ori. În acest caz se recomandă folosirea instrucțiunii FOR. Instrucțiunea FOR are două forme.

Forma 1:

FOR variabilă : = expresie 1 TO expresie 2 DO instrucțiune.

Cuvântul variabilă semnifică variabila de ciclare și ea poate fi de orice tip ordinal. Expresie 1, expresie 2 sunt expresii care au același tip ca și variabila de ciclare.

Principiul de execuție este următorul:

- se evaluează cele două expresii;
- dacă expresia 1 are un ordin mai mare decât expresia 2 se trece mai departe, iar variabila de ciclare are valoarea expresiei 1;
- dacă cele două expresii au aceeași valoare, se execută o singură dată

instrucțiunea, variabila de ciclare ia valoarea comună celor două expresii;
- dacă expresia 1 are un ordin mai mic decât cel al expresiei 2 se procedează astfel:
  - variabila de ciclare ia valoarea expresiei 1 și se execută instrucțiunea;
  - variabila de ciclare ia valoarea succ (din vechea valoare a variabilei de ciclare) și se execută instrucțiunea;
  - procedeul continuă, repetitiv, până când se execută instrucțiunea și pentru variabila de ciclare egală cu valoarea expresiei 2 (în acest caz, după ieșirea din ciclu variabila de ciclare va avea valoarea expresiei 2).

Forma 2:

FOR variabilă ciclare: = expresie 1 **DOWNTO** expresie 2 DO instrucțiune.

Diferența față de forma 1 este că la fiecare pas variabila de ciclare va lua ca valoare pred (vechea valoare).

Exemplul 1:

Să se calculeze suma primelor n numere naturale utilizând instrucțiunea FOR. Se prezintă două programe care calculează această sumă (se utilizează cele două forme ale instrucțiunii FOR).

```
program f1;
var n,s,i:integer;

begin
  write('n=');
  readln(n);
  s:=0;
  for i:=1 to n do s:=s+i;
  writeln('s=',s)
end.

program f2;
var n,s,i:integer;

begin
  write('n=');
  readln(n);
  s:=0;
  for i:=n downto 1 do s:=s+i;
  writeln('s=',s)
end.
```

Exemplul 2:

Să se listeze alfabetul. Se prezintă două variante de listare în ordine normală și inversă.

```
program f3;
var i:char;

begin
  for i:='a' to 'z' do write(i);
```

```
end.

program f4;
var i:char;

begin
  for i:='z' downto 'a' do write(i);
end.
```

Exemplul 3:

Este unul fără efect deosebit, dar demonstrează varietatea tipurilor de date care pot fi folosite de FOR. În acest exemplu se listează primele trei luni ale anului în ordine naturală și în ordine inversă.

```
program f5;
var i:(ianuarie,februarie,martie);

begin
for i:=ianuarie to martie do
  case i of
    ianuarie: writeln('ianuarie');
    februarie: writeln('februarie');
    martie: writeln('martie')
  end
end.

program f6;
var i:(ianuarie,februarie,martie);

begin
for i:=martie downto ianuarie do
  case i of
    ianuarie: writeln('ianuarie');
    februarie: writeln('februarie');
    martie: writeln('martie')
  end
end.
```

Pentru exemplele care urmează, algoritmii au fost prezentați în capitolul 2.

Exemplul 4:

Pentru n citit (n natural mai mare sau egal cu 1) să se calculeze suma $S=1+1*2+1*2*3+...+1*2*...*n$.

```
program p5;
var n,i,s,p:integer;

begin
  write('n=');
  readln(n);
  s:=0;
  p:=1;
  for i:=1 to n do
```

```
  begin
    p:=p*i;
    s:=s+p
  end;
  writeln('s=',s)
end.
```

Exemplul 5:

Se citește n număr natural. Să se precizeze dacă este prim sau nu.

Se prezintă două variante de program care rezolvă această problemă.

```
program p8;
var n,i:integer;
    prim:boolean;

begin
  write('n=');
  readln(n);
  prim:=true;
  for i:=2 to trunc(sqrt(n)) do
    if n mod i =0 then prim:=false;
  if prim  then writeln('numarul este prim')
           else writeln('numarul nu este prim')
end.
```

```
program p8;
var n,i:integer;
    prim:boolean;

begin
  write('n=');
  readln(n);
  prim:=true;
  for i:=2 to trunc(sqrt(n)) do
    if n mod i =0 then prim:=false;
  if prim  then writeln('numarul este prim')
           else writeln('numarul nu este prim')
end.
```

Exemplul 6:

Să se scrie un program care tipărește primele nr numere prime (nr se citește).

```
program p9;
var n,i,j,nr:integer;
    prim:boolean;

begin
  n:=1;
  j:=0;
  write('nr=');
  readln(nr);
  repeat
    prim:=true;
```

```
    for i:=2 to trunc(sqrt(n)) do
      if n mod i=0 then prim:=false;
    if prim then
    begin
      writeln(n);
      j:=j+1
    end;
    n:=n+1
  until j=nr
end.
```

Exemplul 7:

Să se tipărească toate numerele naturale mai mici sau egale cu 10000 care se pot descompune în două moduri diferite ca sumă de cuburi.

```
program p11;
var n,nr,max,i,j,i1,j1,i2,j2:integer;

begin
  for n:=1 to 10000 do
    begin
      max:=trunc(exp(1/3*ln(n)));
      nr:=0;
      for i:=1 to max do
        for j:=i to max do
          if i*i*i+j*j*j=n then
          begin
            if nr=0 then
            begin
              i1:=i;
              j1:=j
            end
            else
            begin
              i2:=i;
              j2:=j
            end;
            nr:=nr+1;
          end;
      if nr>1 then writeln(n,' ',i1,' ',j1,' ',i2,' ',j2)
    end
end.
```

Prezentăm instrucțiunea FOR cu ajutorul diagramelor de sintaxă.

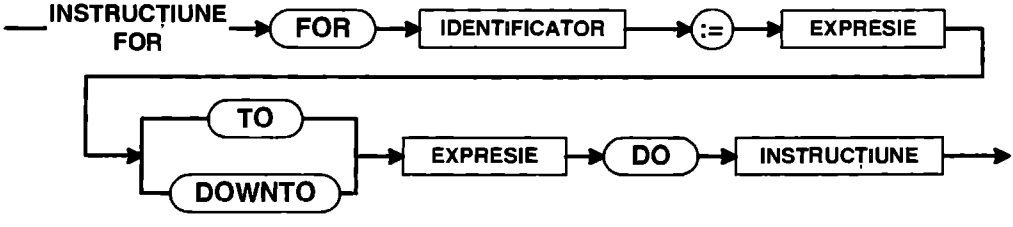


Figura 4.8.

4.9. Aplicaţii la capitolele 3 şi 4

Algoritmi simpli

Pentru aplicaţiile de mai jos, algoritmii se vor redacta utilizând schemele logice, un limbaj de tip pseudocod şi programe PASCAL.

1. Se citeşte o valoare întreagă. Să se precizeze dacă este pozitivă sau nu.

2. Se citesc 3 valori reale a, b, c. Să se precizeze dacă ele pot fi laturile unui triunghi.

3. Să se scrie un program care rezolvă o ecuaţie de gradul 2.

4. Se citesc 4 numere întregi. Să se tipărească în ordine descrescătoare.

5. Să se calculeze valoarea funcţiei F, definită pe mulţimea numerelor reale, pentru un x citit:

$$F = \begin{cases} x*x-3, & x<5 \\ x+1, & 5 \le x \le 25 \\ x*x-5*x+6, & x>25 \end{cases}$$

6. Se citesc 4 valori reale a, b, c, d. Să se evalueze expresia:

$$E = \begin{cases} a+b, & c+d>0 \\ a-b, & c+d=0 \\ a*b, & c+d<0 \end{cases}$$

7. Se citesc a, b, c, coeficienţii unei ecuaţii de gradul 2. Fără a rezolva ecuaţia, să se precizeze natura rădăcinilor (reale sau nu, pozitive, negative, semnul rădăcinilor (dacă sunt reale).

8. Se citesc n numere reale. Să se tipărească valoarea minimă citită.

9. Se citeşte un şir de numere întregi până la întâlnirea numărului 0. Să se calculeze media aritmetică a numerelor din şir.

10. Citim un şir de numere întregi, la fel ca la problema precedentă. Să se calculeze suma dintre primul număr, al treilea, al cincilea,... şi produsul dintre al doilea, al patrulea,... etc.

11. Se citesc 3 numere naturale n, p, k şi un şir de n numere naturale. Câte dintre acestea, împărţite la p dau restul k?

12. Să se calculeze expresiile de mai jos (programe separate):

$$E_1 = 1 + \frac{1}{2^2} + \frac{1}{3^2} + \ldots + \frac{1}{n^2}$$

$$E_2 = \frac{1}{2 \cdot 3} + \frac{2}{3 \cdot 4} + \ldots + \frac{n}{(n+1)(n+2)}$$

$$E_3 = 1 + \frac{1}{1!} + \frac{1}{2!} + \ldots + \frac{1}{n!}, \qquad n! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot n$$

13. Să se calculeze produsul a două numere naturale prin adunări repetate.

14. Efectuați împărțirea întreagă a două numere citite, fără a utiliza operatorii MOD și DIV.

15. Se citește o succesiune de cifre 1 și 0 în care prima este 1. Aceasta are semnificația unui număr în baza 2. Să se tipărească numărul în baza 10.

16. Se citește o succesiune de caractere (într-o variabilă de tip char) '0', '1', ..., '9', 'a', 'b', ..., 'f' în care primul caracter citit nu este 0. Aceasta are semnificația unui număr în baza 16. Să se tipărească numărul în baza 10.

17. Se consideră șirul definit astfel (FIBONACCI):

$$u(n) = \begin{cases} 1, & \text{dacă } n=1 \text{ sau } n=2 \\ u(n-1)+u(n-2), & \text{altfel} \end{cases}$$

Se citește o valoare naturală a. Să se calculeze u(a).

18. Se consideră șirul definit la problema anterioară. Să se tipărească numărul lui de termeni care sunt strict mai mici decât o valoare citită.

19. Se citesc două numere naturale m și n. Să se calculeze cel mai mare divizor comun al lor:

a) prin utilizarea algoritmului lui EUCLID;

b) prin utilizarea relației de mai jos:

$$cmmdc(a,b) = \begin{cases} cmmdc(a-b,b), & \text{dacă } a>b; \\ cmmdc(a,b-a), & \text{dacă } a<b; \\ a, & \text{dacă } a=b. \end{cases}$$

20. Se citește un număr natural. Câte cifre conține?

21. Calculați suma și produsul divizorilor primi ai unui număr citit.

22. Se citesc, pe rând, cifrele unui număr (începând cu cifra cea mai semnificativă). Să se formeze numărul, într-o variabilă de tip integer.

23. Se citește un număr cu 5 cifre. Tipăriți numărul format după eliminarea cifrei din mijloc.

24. Se citește un număr cu 8 zecimale (partea întreagă a acestuia este 0).

Să se tipărească numărul rezultat prin eliminarea primelor două și ultimelor două zecimale. Exemplu: 0.12345678, se va tipări 0.3456.

25. În practică, un rol fundamental îl joacă numerele aleatoare (întâmplătoare). Există mulți algoritmi care permit generarea lor, însă aceștia implică niște cunoștințe de matematică ce depășesc cu mult programa de liceu. Aici vom prezenta pe cel mai modest dintre ei, dar și cel mai simplu. Se consideră un număr cu 8 zecimale și se înlătură 4 dintre acestea, întocmai ca la problema anterioară. Numărul obținut se ridică la pătrat. În acest fel obținem un nou număr cu 8 zecimale, cu care se procedează în mod identic. Citiți un număr cu 8 zecimale și tipăriți primele n numere obținute prin procedeul indicat.

Observație: pentru un șir astfel obținut, la un moment dat valorile se repetă (se spune că șirul este periodic). În acest sens, pentru ca șirul să se repete după un număr cât mai mare de valori, nu este indiferent numărul de la care se pleacă. Încercați programul stabilit, pentru mai multe numere de pornire, inclusiv pentru cel definit în problema 24.

26. Pornind de la problema anterioară, găsiți un procedeu de generare a numerelor aleatoare întregi.

27. Găsiți un procedeu de generare a unor numere aleatoare întregi care să se găsească între două valori întregi citite.

28. Citiți un număr natural n și unul real $R<1$; generați n perechi de numere aleatoare de forma (x,y), cu x și y valori pozitive, cuprinse în intervalul [0,R). Câte perechi dintre acestea verifică relația $x*x+y*y<R*R$?

Dacă reprezentăm în plan un punct de coordonate (x,y), unde x și y verifică relația de mai sus, acesta se va găsi în interiorul unei figuri determinate de axa OX (în sens pozitiv), axa OY (în sens pozitiv) și cercul de rază R, cu centrul în origine. Aria acestei figuri este $\pi*R*R/4$. Indiferent dacă perechea (x,y) verifică sau nu relația de mai sus, punctul se va găsi în interiorul unui pătrat de latură R (aria R*R). În cazul în care punctele se distribuie proporțional în interiorul unei figuri, aria figurii este proporțională cu numărul punctelor care se găsesc în interiorul ei. În concluzie, raportul celor două arii considerate este aproximativ egal cu raportul dintre numărul punctelor care se găsesc în interiorul figurii mai sus precizate și numărul total de puncte (care sunt oricum în interiorul pătratului). Raportul celor două arii este $\pi/4$.

Calculați câtul dintre numărul de puncte care se găsesc în interiorul cercului și numărul total de puncte, înmulțiți rezultatul cu 4. În acest fel, se obține o valoare aproximativă pentru numărul π. Încercați algoritmul cu diferite numere de pornire (pentru generare) și cu un număr variabil de puncte care vor fi reprezentate.

29. Se consideră un triunghi echilateral înscris într-un cerc de rază R. Care este lungimea laturii triunghiului echilateral în funcție de raza cercului? Unim mijlocul M al unei laturi a triunghiului cu centrul cercului O și prelungim segmentul OM până intersectează cercul într-un punct P. Se trasează segmentele care unesc extremitățile laturii cu punctul P. Procedând în mod asemănător cu fiecare latură, se obține un hexagon regulat (de ce?). Cu același procedeu, acestuia i se dublează numărul de laturi ș.a.m.d.

acestuia i se dublează numărul de laturi ș.a.m.d.

- Considerând *ln* lungimea laturii unui poligon regulat, înscris într-un cerc de rază R, să se calculeze *l2n*, lungimea laturii poligonului regulat cu un număr dublu de laturi, înscris în același cerc și care a fost obținut așa cum s-a arătat mai sus (se va calcula *l2n* în funcție de *ln* și R, fără a utiliza funcții trigonometrice).
- În cazul în care se pornește de la triunghiul echilateral, se dublează numărul de laturi, iar se dublează (de n ori) și se obține un poligon regulat al cărui perimetru se cere.
- După un număr de astfel de dublări ale laturilor, perimetrul poligonului regulat care se obține aproximează destul de bine lungimea cercului ($2*\pi*R$). În concluzie, dacă se împarte perimetrul la $2*R$, se obține o valoare aproximativă a numărului π.

Scrieți un program care să calculeze astfel numărul π. Programul va avea ca dată de intrare numărul natural n (de câte ori dublăm laturile triunghiului echilateral).

Testați programul pentru diverse valori ale lui n. Veți observa că, pentru valori mai mari ale lui n, vom obține pentru π valoarea 0 (absurd). Explicați care este cauza.

30. Un număr este perfect dacă este egal cu suma divizorilor săi (excluzându-l pe el). Exemplu: $6=1+2+3$. Tipăriți toate numerele perfecte până la un întreg citit.

31. Până la un întreg citit, tipăriți toate numerele naturale care sunt egale cu suma pătratelor cifrelor lor.

32. Determinați numărul de zile trecute între două date calendaristice citite (anii bisecți sunt cei divizibili cu 4).

33. Afișați primele n perechi de numere prime care sunt consecutive în mulțimea numerelor impare. Exemplu: (3,5), (5,7).

34. Se citește un număr natural par. Să se descompună în sumă de numere prime (conjectura GOLDBACH).

35. Se citesc două numere naturale formate din câte 4 cifre distincte. Care este numărul cifrelor comune celor două numere (nu coincid obligatoriu și ca poziție)?

36. Se citesc 2 numere naturale n și m. Calculați n la puterea m efectuând, pe cât posibil, mai puțin de $m-1$ înmulțiri.

37. Se citesc n numere naturale. Să se tipărească cel mai mare divizor comun al lor.

38. Se citesc a, b, c coeficienții unei ecuații de gradul 2 și un număr natural n. Fără a rezolva ecuația să se calculeze expresia:

$$S_n = x_1^n + x_2^n$$

Indicație:

x_1 rădăcină $\Rightarrow ax_1^2 + bx_1 + c = 0$

Înmulțim relația cu $x_1^n \Rightarrow ax_1^{n+2} + bx_1^{n+1} + cx_1^n = 0$.

Analog, $ax_2^{n+2} + bx_2^{n+1} + cx_2^n = 0$.

Prin adunare $\Rightarrow a(x_1^{n+2} + x_2^{n+2}) + b(x_1^{n+1} + x_2^{n+1}) + c(x_1^n + x_2^n) = 0$

$\Rightarrow aS_{n+2} + bS_{n+1} + cS_n = 0$, dar $S_1 = -\frac{b}{a}$, $S_2 = x_1^2 + x_2^2 = (x_1 + x_2)^2 - 2x_1x_2 = \frac{b^2}{a^2} - 2\frac{c}{a}$.

Programați o relație de recurență.

39. Rezolvați în mulțimea numerelor întregi ecuația:

$x * x - y * y = k$ (k natural);

Indicație:

Putem scrie: $(x - y) * (x + y) = k$.

Notăm: $x - y = a$; $x + y = b$;

Rezultă: $x = (a + b) / 2$; $y = (b - a) / 2$.

Pe altă parte, a și b trebuie să fie divizori ai lui k. De asemenea se ține cont de faptul că, dacă ecuația admite soluția (x,y) admite și soluțiile (–x, –y), (–x, y), (x, –y).

40. Se citesc pe rând n numere naturale și un număr p. Se cere să se găsească k maxim, astfel încât p la puterea k divide produsul celor n numere naturale. Se va evita efectuarea produsului celor n numere naturale.

Capitolul 5

Tipuri structurate de date

5.1. Tipul tablou

5.1.1. Tipul ARRAY

Pentru explicarea necesității **tipului ARRAY** vom porni de la o aplicație practică. Se citesc 100 de numere naturale. Să se precizeze dacă acestea sunt distincte sau nu. Pentru rezolvare, în program ar trebui să reținem o sută de variabile de tip întreg. Pe lângă acest dezavantaj major, imaginați-vă cum ar arăta secvența de program care testează dacă numerele conținute de cele 100 de variabile sunt distincte sau nu. Dar dacă ar trebui să rezolvăm această problemă pentru 1000 de numere? Șirul de exemple care să justifice utilizarea tipului ARRAY ar putea continua la nesfârșit.

Cum se declară variabilele de acest tip?

Programul următor (care de altfel nu realizează nimic) exemplifică astfel de declarații.

```
program a1;
type  vector=array [1..100] of integer;
     vector1=array [-10..10] of real;
var a:vector;
     b,c:vector1;
     d:array[1..21] of char;
begin
end.
```

O variabilă de tip vector va conține 100 de componente. Fiecare componentă va reține un număr întreg. O variabilă de tip vector1 conține 21 de componente. Fiecare din ele reține un număr real. Variabila d are 21 de componente care memorează câte un caracter.

Observații:

- Nu putem să declarăm un tip array cu un număr variabil de componente (așa cum este permis în limbajul BASIC, de exemplu: citim n și declarăm o variabilă care are n componente).
- Nu este obligatoriu ca **indicele** componentelor să fie cuprins între 1 și n (tipul vector1 are și componente de indice negativ). Prin indicele unei componente înțelegem valoarea prin care putem adresa componenta.
- Se pot declara variabile de un tip **anonim** (tipul variabilei d a fost precizat la declararea variabilei).

- În memoria calculatorului componentele unei variabile de tip array se reţin una după alta (fără a exista spaţiu între ele).

Cum se adresează componentele unei variabile de tip array?
Pentru adresarea unei componente, se foloseşte numele variabilei urmat de indicele ei. Indicele este trecut în mod obligatoriu între paranteze drepte.

Programul care urmează citeşte şi tipăreşte componentele unei variabile de tip array. În final, se tipăreşte din nou conţinutul componentei a doua.

```
program a2;
type vector=array [1..100] of integer;
var a:vector;
    n,i:integer;
begin
write('n=');
readln(n);
for i:=1 to n do   begin
                   write('a[',i,']=');
                   readln(a[i])
                   end;
for i:=1 to n do writeln('a[',i,']=',a[i]);
writeln(a[2])
end.
```

Observaţii:
- Este indicat ca la citirea unei componente să se tipărească şi numărul ei de ordine (pentru ca acela care introduce datele să ştie de fiecare dată ce componentă introduce);
- Nu este obligatoriu ca indicele să fie conţinut într-o variabilă. El poate fi trecut printr-o valoare (am procedat astfel când am tipărit componenta doi).

Teoretizând, tipul array se declară după diagrama de sintaxă de mai jos:

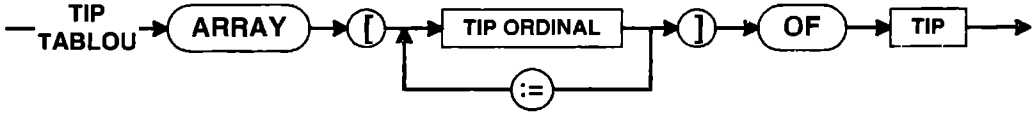


Figura 5.1.1.

După cum rezultă din diagramă, indicele poate fi de un tip ordinal (cu un număr finit de elemente). Nu poate fi indice un tip INTEGER dar este admis pentru indici un subdomeniu al tipului INTEGER aşa cum de altfel am folosit în toate exemplele de până acum.
În diagramă, prin TIP se înţelege tipul de bază al tabloului, adică tipul componentelor acestuia.

Să exemplificăm cele prezentate în diagramă, arătând astfel şi extraordinara diversitate a tipurilor de date ale limbajului. Programul care urmează citeşte şi tipăreşte o variabilă cu componente de tip întreg, dar indicele componentelor este tipul char (care este un tip ordinal).

```
program a3;
type vector=array [char] of integer;
var a:vector;
    i:char;

begin
  for i:='a' to 'c' do
  begin
    write('a[',i,']=');
    readln(a[i])
  end;
  for i:='a' to 'c' do writeln('a[',i,']=',a[i]);
end.
```

Programul a4 citește o variabilă de tip array cu componente reale și indice de tip enumerare.

```
program a4;
type  zile=(luni,marti,miercuri,joi);
    vector=array [zile] of real;
var a:vector;
    i:zile;

begin
  for i:=luni to joi do readln(a[i]);
  for i:=luni to joi do writeln(a[i]:5:2);
end.
```

Componentele unei variabile de tip array pot fi la rândul lor variabile de tip array.

În programul următor tipul matrice are trei componente de tip vector. Programul citește și tipărește o variabilă de **tip matrice**. Observați modul de adresare a componentelor.

```
program a5;
type  vector=array[1..2] of integer;
    matrice=array[1..3] of vector;
var i,j:integer;
    a:matrice;

begin
  for i:=1 to 3 do
    for j:=1 to 2 do
      readln(a[i][j]);
  for i:=1 to 3  do
    for j:=1 to 2 do
      writeln(a[i][j])
end.
```

Modul de adresare este depărtat de cel folosit în matematică. Astfel, în matematică o matrice este un tablou cu un număr de linii și coloane. Într-o anumită linie și coloană se găsește un element de un anumit tip (întreg, real ș.a). O matrice arată așa cum se vede mai jos:

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix}$$

Elementul din linia i și coloana j se simbolizează prin a(i,j), unde a reprezintă matricea. Pentru a putea scrie cu ușurință programe care lucrează cu matrice, limbajul Pascal permite și o altfel de adresare pentru componentele de acest tip. Pentru un astfel de tip se folosesc doi indici de adresare. Fiecare din ei trebuie să respecte regulile prezentate pentru un singur indice. Programul care urmează prezintă citirea și scrierea unei matrice cu trei linii și două coloane (de același tip cu cea folosită în programul anterior).

```
program a5;
type matrice=array[1..3,1..2] of integer;
var i,j:integer;
    a:matrice;

begin
  for i:=1 to 3 do
    for j:=1 to 2 do
      readln(a[i,j]);
  for i:=1 to 3 do
    for j:=1 to 2 do
      writeln(a[i,j])
end.
```

Este momentul să clarificăm anumite probleme care apar atunci când se lucrează cu tipul array. Dacă un astfel de tip are componente care nu sunt de tip array spunem că lucrăm cu vectori.

În situația în care componentele sunt de tip array și la rândul lor au componente care nu sunt de tip array spunem că lucrăm cu matrice. Vectorii au o singură dimensiune, iar matricele au două dimensiuni (linie, coloană). Evident și matricele se pot generaliza în sensul că acestea au componente de tip vector. Pentru a identifica un element al unui astfel de tip avem nevoie de trei indici, motiv pentru care spunem că astfel de structuri au trei dimensiuni. În general, putem lucra cu structuri n dimensionale.

Aplicații:

Problema 1

Se citește un vector cu n componente (numere naturale). Se cere să se tipărească valoarea maximă dintre numerele citite.

Algoritmul de rezolvare este următorul:

- variabilei max i se atribuie valoarea primei componente;
- pe rând, se compară valoarea variabilei max cu valorile existente în componenta 2, 3, ..., n și de câte ori valoarea reținută într-o componentă este mai mare decât valoarea reținută în max, variabilei max i se atribuie

valoarea acelei componente.
Acest algoritm efectuează n–1 comparații.

```
program m;
type vector=array [1..9] of integer;
var v:vector;
    i,max,n:integer;

begin
  write('n=');
  readln(n);
  for i:=1 to n do
  begin
    write('v[ ',i,']=');
    readln(v[i])
  end;
  max:=v[1];
  for i:=2 to n do
    if v[i]>max then max:=v[i];
  writeln('max=',max)
end.
```

Exercițiu: Scrieți programul care calculează minimul dintre componentele citite ale unui vector.

Problema 2
Se citește un vector având n componente numere întregi. Se cere să se decidă dacă numerele citite sunt distincte sau nu.

Principiul de rezolvare a acestei probleme este următorul:
- se compară valoarea primei componente cu valorile componentelor 2, 3,..., n;
- se compară valoarea componentei doi cu valorile componentelor 3, 4,..., n;
- se compară valoarea componentei n–1 cu valoarea componentei n.

În cazul în care în toate aceste comparații nu se găsește nici o egalitate, rezultă că numerele citite sunt distincte. La prima egalitate găsită rezultă că numerele nu sunt distincte și programul se termină cu mesajul corespunzător.

Analizați modul în care a fost scris programul astfel încât la prima egalitate găsită să se oprească.

```
program dist;
type vector=array [1..9] of integer;
var a:vector;
    gasit:boolean;
    n,i,j:integer;

begin
  write('n=');
  readln(n);
  for i:=1 to n do
  begin
    write('a[',i,']=');
    readln(a[i])
  end;
```

```
    gasit:=false;
    i:=1;
    while not gasit and(i<=n) do
    begin
      j:=i+1;
      while (j<=n) and not gasit do
      begin
        if a[i]=a[j] then gasit:=true;
        j:=j+1
      end;
      i:=i+1
    end;
    if gasit then writeln('elementele nu sunt distincte')
             else writeln('elementele sunt distincte')
end.
```

Dacă numerele reținute de un vector sunt distincte, rezultă că ele pot alcătui o mulțime. Din acest motiv, un astfel de test se mai numește și test de mulțime.

Problema 3

Se citesc doi vectori a și b cu componente numere naturale distincte (datele introduse de operator se presupun a fi distincte). Cei doi vectori au n, respectiv m componente și au semnificația de mulțimi. Se cere să se construiască vectorul c care reprezintă reuniunea celor două mulțimi.

Algoritmul de rezolvare este următorul:

- se copiază componentele vectorului a în vectorul c (reuniunea include oricare din mulțimile din care rezultă);
- se adaugă vectorului c acele componente ale vectorului b care nu se regăsesc în vectorul a;
- se tipărește conținutul vectorului c.

Observații:

- Variabila k arată componenta vectorului c în care urmează să se scrie acea componentă a vectorului b care nu este egală cu nici o componentă a vectorului a. Care este motivul pentru care, în final, se listează primele k−1 componente ale vectorului c?
- Se presupune că vectorul c nu va avea mai mult de 9 componente (vezi declararea tipului vector).

```
program reun;
type vector=array [1..9] of integer;
var a,b,c:vector;
    n,m,i,j,k:integer;
    gasit:boolean;

begin
  writeln('multimea A');
  write('n=');
  readln(n);
  for i:=1 to n do
  begin
    write('a[',i,']=');
```

```
    readln(a[i])
  end;
  writeln('multimea B');
  write('m=');
  readln(m);
  for i:=1 to m do
  begin
    write('b[',i,']=');
    readln(b[i])
  end;
  for i:=1 to n do c[i]:=a[i];
  k:=i+1;
  for i:=1 to m do
  begin
    gasit:=false;
    j:=1;
    while not gasit and (j<=n) do
    begin
      if a[j]=b[i] then gasit:=true;
      j:=j+1
    end;
    if not gasit then
    begin
      c[k]:=b[i];
      k:=k+1
    end
  end;
  writeln('reuniunea');
  for i:=1 to k-1 do writeln(c[i]);
end.
```

Problema 4

Se citesc doi vectori, a și b, cu n și m componente numere naturale, având semnificația de mulțimi. Se cere să se construiască și tipărească vectorul c, care reprezintă intersecția celor două mulțimi.

Pornind de la faptul că intersecția a două mulțimi este formată din elementele comune celor două mulțimi, vectorul c se construiește din acele elemente ale vectorului a care sunt și elemente ale lui b.

Care este motivul pentru care căutarea unui element al mulțimii a în mulțimea b nu se face utilizând instrucțiunea for?

```
program inters;
type vector=array [1..9] of integer;
var a,b,c:vector;
    n,m,i,j,k:integer;
    gasit:boolean;

begin
  writeln('multimea A');
  write('n=');
  readln(n);
  for i:=1 to n do
  begin
    write('a[',i,']=');
```

```
    readln(a[i])
  end;
  writeln('multimea B');
  write('m=');
  readln(m);
  for i:=1 to m do
  begin
    write('b[',i,']=');
    readln(b[i])
  end;
  k:=0;
  for i:=1 to n do
  begin
    gasit:=false;
    j:=1;
    while not gasit and (j<=m) do
    begin
      if a[i]=b[j] then gasit:=true;
      j:=j+1
    end;
    if gasit then
    begin
      k:=k+1;
      c[k]:=a[i]
    end
  end;
  writeln ('intersectia ');
  for i:=1 to k do  writeln(c[i])
end.
```

Problema 5

Se citește un vector cu n componente numere naturale. Se cere să se sorteze componentele acestui vector în ordine descrescătoare.

Există foarte mulți algoritmi de sortare. Aici vom considera algoritmul cel mai simplu, acela la care s-ar gândi cineva care este pus pentru prima dată în fața acestei probleme. Aceasta nu înseamnă că acest algoritm este și eficient, dimpotrivă.

În ce constă acesta? Se calculează maximul dintre elementele vectorului. Acesta este trecut pe prima poziție a unui alt vector numit vs și care va conține componentele sortate. Dintre componentele rămase se calculează din nou maximul care este trecut pe poziția 2 a vectorului vs. Procedeul continuă până când sunt trecute toate componentele vectorului care se sortează în vs. În final, se listează vs. Pentru a calcula mereu maximul dintre componentele rămase, se consideră un vector suplimentar, numit s, care are tot n componente. Atât timp cât componenta i a vectorului v nu a fost găsită maximă și deci selectată, s(i) are valoarea 1, în caz contrar are valoarea 0.

```
program sort1;
type vector=array[1..9] of integer;
var v,vs,s:vector;
    n,i,j,k,max,poz:integer;

begin
```

```
    write('n=');
    readln(n);
    for i:=1 to n do
    begin
      write('v[',i,']=');
      readln(v[i])
    end;
    for i:=1 to n do s[i]:=1;
    for i:=1 to n do
    begin
      j:=1;
      while s[j]=0 do j:=j+1;
      if j<=n then
      begin
        max:=v[j];
        poz:=j;
        for k:=j+1 to n do
          if s[k]=1 then
            if v[k]>max then
            begin
              max:=v[k];
              poz:=k
            end;
        s[poz]:=0;
        vs[i]:=max
      end;
    end;
    for i:=1 to n do writeln(vs[i]);
  end.
```

Problema 6

Se citește un vector cu n componente numere întregi. Să se sorteze crescător.

Desigur, se poate folosi algoritmul din problema precedentă cu deosebirea că aici se va calcula de fiecare dată minimul. Vom folosi o altă metodă, numită **metoda bulelor**. În ce constă ea? Se parcurge vectorul de mai multe ori și se inversează de fiecare dată elementele alăturate care nu respectă relația de ordine considerată. Procedeul continuă până când la o parcurgere a vectorului nu am făcut nici o inversare.

Exemplu: Să presupunem că am citit 4, 1, 3, 2.

- se parcurge pentru prima dată vectorul:
  - se inversează 4 cu 1 și se obține 1, 4, 3, 2;
  - se inversează 4 cu 3 și se obține 1, 3, 4, 2;
  - se inversează 4 cu 2 și se obține 1, 3, 2, 4;
- întrucât la această parcurgere am avut inversări, reparcurgem vectorul:
  - se inversează 3 cu 2 și obținem 1, 2, 3, 4;
- întrucât am avut inversări, se reparcurge vectorul:
  - nu se face nici o inversare;
- în acest moment vectorul este sortat, iar algoritmul se oprește.

Observați că ideea de sortare este mai greu de găsit, dar algoritmul acesta este mult mai simplu decât cel anterior.

Care este modificarea ce trebuie făcută astfel încât vectorul să fie sortat descrescător?

```
program sort2;
type vector=array[1..10] of integer;
var n,i,m:integer;
    v:vector;
    inversari:boolean;

begin
  write('n=');
  readln(n);
  for i:=1 to n do
  begin
    write('v[',i,']=');
    readln(v[i])
  end;
  repeat
    inversari:=false;
    for i:=1 to n-1 do
      if v[i]>v[i+1] then
      begin
        m:=v[i];
        v[i]:=v[i+1];
        v[i+1]:=m;
        inversari:=true
      end;
  until not inversari;
  writeln('vectorul sortat ');
  for i:=1 to n do writeln(v[i])
end.
```

5.1.2. Tipul STRING

Se propune următoarea problemă: să se citească și să se tipărească cuvântul 'calculator'. Pentru aceasta, trebuie să procedăm astfel:

- reținem un vector cu cel puțin 10 componente de tip char;
- citim cuvântul, caracter cu caracter;
- îl tipărim (paradoxal, tipărirea se poate face în ansamblu).

Programul următor realizează toate acestea:

```
program st1;
var a:array[1..10] of char;
   i:integer;

begin
  for i:=1 to 10 do read(a[i]);
  writeln;
  writeln(a);
end.
```

Să recunoaștem că o astfel de modalitate de lucru este deosebit de greoaie. Dar dacă citim un cuvânt cu patru litere? S-ar putea scrie o secvență care să citească cuvântul până la întâlnirea caracterului ENTER, însă și această modalitate este dificilă.

Limbajul Turbo Pascal conține un tip de date, numit **STRING**, care permite să se lucreze cu mare ușurință în aceste cazuri. Iată cum arată programul care rezolvă problema pusă la începutul acestui paragraf:

```
program st2;
var a:string;

begin
  readln(a);
  writeln(a);
end.
```

O astfel de rezolvare a problemei propuse prezintă și un alt avantaj și anume, se poate citi un cuvânt (sau mai multe cuvinte separate prin blancuri) fără a ști de la început numărul de caractere ce vor fi citite (totuși acest număr trebuie să fie mai mic sau egal cu 255).

O variabilă de tip STRING poate reține maximum 255 de caractere și pentru ea se rezervă în memorie 256 de octeți (în primul octet se reține lungimea efectivă a cuvântului citit).

Mai mult, nu suntem obligați să declarăm variabile de acest tip care să conțină 256 de octeți. Există posibilitatea să declarăm variabile de tip STRING cu un număr mai mic de octeți. De exemplu, dacă la problema propusă știm de la început că lungimea cuvântului citit nu depășește 20 de caractere, putem proceda ca în programul următor:

```
program st3;
var a:string[20];

begin
  readln(a);
  writeln(a);
end.
```

De această dată, pentru variabila a s-au reținut 21 de octeți.

Ca și la tipul array, există posibilitatea de a adresa un anumit caracter pentru cuvântul citit. De exemplu, dacă variabila a conține cuvântul 'text', a[2] va fi 'e'. Avem posibilitatea de a cunoaște lungimea efectivă a cuvântului prin numărul conținut în octetul a[0] (pentru exemplul nostru, acesta va conține numărul 4).

```
program st4;
var a:string;

begin
  readln(a);
  writeln(a);
```

```
  writeln(a[2]);
  writeln('cuvântul citit are ',ord(a[0]), ' litere')
end.
```

Variabilele de tip STRING pot face obiectul instrucțiunii de atribuire.

Considerăm instrucțiunea a:=b, unde a și b sunt variabile de tip string. În principiu, în urma execuției acestei instrucțiuni variabila b are același conținut ca și variabila a.

Distingem, totuși mai multe situații.

- Dacă variabila a are rezervați mai puțini octeți decât conținutul efectiv al variabilei b, șirul de caractere ce va fi conținut de a va avea un număr de caractere egal cu numărul caracterelor pe care le poate reține a și anume primele poziții de la stânga la dreapta ale șirului conținut în b (nu se semnalează eroare).
- În situația în care variabila a are rezervați mai puțini octeți decât variabila b, dar șirul conținut de b are un număr de caractere mai mic sau egal cu numărul octeților rezervați pentru a, atribuirea se face normal (a va avea conținutul lui b).

Testați acestea, folosind programul următor:

```
program st5;
var a:string[10];
    b:string[4];
    c:string;

begin
  readln(b);
  a:=b;
  writeln(a);
  readln(a);
  b:=a;
  writeln(b);
  a:='text ';
  b:='baza';
  c:=a+b;
  writeln(c)
end.
```

Programul prezentat conține și atribuirea c:=a+b. În cazul variabilelor de acest tip operația **+** are semnificația de **concatenare**. Variabila c va avea drept conținut șirul de caractere ce rezultă din șirul conținut de a, urmat de șirul conținut de b. În program, c va avea conținutul 'text baza'.

Variabilelor de tip STRING li se pot aplica **operatori relaționali**. Aceștia sunt: **=** (egal), **< >** (diferit), **<** (mai mic), **<=** (mai mic sau egal), **>** (mai mare), **>=** (mai mare sau egal).

Care este mecanismul prin care se realizează comparația? Cum este posibil ca un cuvânt să fie (de exemplu) mai mare decât altul? Să nu uităm că fiecare caracter are un anumit cod (un număr care se reține pe un octet). Codurile nu au fost alese întâmplător, ci a fost respectată ordinea alfabetică a caracterelor (de exemplu caracterul b are un cod mai mare decât caracterul a). Comparația se realizează, de fapt, prin codurile caracterelor. Să presupunem că se compară două șiruri, primul cu m caractere, al doilea cu n caractere (m<=n).

Se procedează astfel:

- se compară codul primului caracter al primului șir cu codul primului caracter al celui de-al doilea șir, în urma comparației se poate ajunge într-una din situațiile următoare:
- primul cod este mai mare, caz în care se consideră primul șir mai mare decât celălalt;
- primul cod este mai mic, astfel că primul șir este mai mic;
- cele două coduri sunt egale (avem același caracter), se trece la comparațiile de cod pentru al doilea caracter al primului șir cu al doilea caracter pentru șirul al doilea și în urma comparației se procedează la fel ca pentru primul caracter.

Presupunem că în urma comparației primelor m caractere ale celor două șiruri am avut tot timpul egalitate. Dacă m = n, rezultă că cele două șiruri sunt egale, iar dacă m < n rezultă că primul șir este mai mic decât al doilea.

Programul st6 tipărește de două ori TRUE.

```
program st6;
var a: string[10];
    b: string[3];

begin
  a:='act';
  b:='act';
  writeln(a=b);
  a:='ma';
  b:='ma1';
  writeln(a<b);
end.
```

Faptul că două șiruri pot fi comparate reprezintă un avantaj enorm în practică. Astfel, putem sorta alfabetic un șir de cuvinte. Programul st7 sortează alfabetic componentele unui vector cu elemente de tip string (deci fiecare element va conține un cuvânt).

```
program st7;
type vector=array[1..10] of string[10];
var man:string[10];
    inv:boolean;
    n,i:integer;
    v:vector;

begin
  write('n=');
  readln(n);
  for i:=1 to n do
  begin
    write('v[',i,']=');
    readln(v[i])
  end;
  repeat
    inv:=false;
    for i:=1 to n-1 do
      if v[i]>v[i+1] then
```

```
      begin
        man:=v[i];
        v[i]:=v[i+1];
        v[i+1]:=man;
        inv:=true
      end
  until not inv;
  for i:=1 to n do
  writeln(v[i])
end.
```

Există mai multe proceduri și funcții care pot lucra cu variabile de tip string. Până în acest moment, noi nu am studiat nici procedurile, nici funcțiile. Acestea se vor studia ulterior. Pentru moment, înțelegem că acestea reprezintă secvențe de program, scrise de alții, care efectuează anumite calcule și returnează anumite rezultate. Rezultatele funcțiilor le regăsim prin numele lor (la fel cum conținutul unei variabile se obține referind variabila prin numele ei) iar rezultatele procedurilor le regăsim conținute în anumite variabile pe care le transmitem atunci când apelăm procedurile. La apel, atât pentru proceduri cât și pentru funcții, se transmit (de cele mai multe ori) ca parametri variabile ale programului nostru. Lucrurile vor fi mai bine lămurite, prin exemple, iar în acest an vom învăța pe larg despre proceduri și funcții.

5.1.2.1. Functia COPY

Se presupune următoarea problemă, care apare de multe ori în practică. Avem o variabilă de tip string ce conține un șir cu m caractere. Se cere să se găsească subșirul care începe în poziția i< =m și care are n caractere. De exemplu, variabila a conține șirul 'calculator' și se dorește să extragem subșirul care începe de la poziția 3 și are 4 caractere. Această problemă se poate rezolva în felul următor:

```
program st8;
var a,b:string;
    i:integer;

begin
  a:='calculator';
  b:='';
  for i:=3 to 6 do b:=b+a[i];
  writeln(b)
end.
```

În program, apare atribuirea b:=''. Aceasta înseamnă că variabilei b i se atribuie șirul vid. În continuare variabilei b i se adaugă pe rând caracterele cuprinse între 3 și 6. Se va tipări cuvântul 'lcul'.

Același rezultat se obține și prin apelarea funcției COPY, cum se vede mai jos:

```
program st9
var a,b: st ing;
```

```
begin
  a:='calculator';
  b:=copy(a,3,4);
  writeln(b)
end.
```

Funcția COPY extrage un subșir dintr-un șir dat. La apel, funcția COPY are trei parametri care trebuie scriși în ordinea de prezentare:

- șirul din care se face extragerea;
- poziția primului caracter al șirului extras;
- numărul de caractere care se extrag.

Observații:

- Este mult mai bine să lucrăm utilizând această funcție datorită faptului că, prin parametri convenabil aleși, se poate extrage subșirul dorit din orice poziție pe orice lungime.
- În exemplu am folosit valori efective pentru poziție și număr de caractere, dar se pot folosi și variabile (acest lucru este adevărat în general pentru orice procedură și funcție). Dacă avem m = 3, n = 4 (m, n variabile de tip integer) putem pune COPY(a,m,n).
- În situația în care lungimea subșirului cerut este mai mare decât numărul de caractere existente în șir începând din poziția solicitată, se extrag numai caracterele din poziția curentă până la sfârșit. *Exemplu:* variabila a conține cuvântul 'tasta'. Funcția Copy(a,3,5) returnează șirul 'sta'.

5.1.2.2. Funcția POS

Se consideră două variabile a și b (de tip string) și fiecare din ele conține un șir de caractere. Se cere să se precizeze dacă șirul conținut de variabila b este sau nu subșir al variabilei a, iar în cazul unui răspuns afirmativ să se precizeze poziția de început a subșirului conținut în b, în șirul conținut în a.

Exemple:

- a conține 'calculator' iar b cuvântul 'lat' . Răspunsul este afirmativ iar poziția de început este 6.
- a conține 'calculator' iar b cuvântul 'lar'. Răspunsul este negativ.

Programul st10 caută subșirul conținut în b în cadrul șirului conținut în a. Dacă este găsit, se tipărește poziția de început a subșirului, în caz contrar se tipărește 0.

```
program st10;
var a,b:string;
    i:integer;

begin
  a:='calculator';
  b:='lat';
  i:=1;
  while (i< length(a)-length(b)+1) and
        (b< copy(a,i,length(b))) do i:=i+1;
```

```
  if i>length(a)-length(b)+1 then i:=0;
  writeln(i)
end.
```

Același lucru, dar mai general și mai simplu se realizează prin utilizarea **funcției POS**. Aceasta are doi parametri: subșirul căutat și șirul unde se caută. Rezultatul este de tip byte. Dacă rezultatul este 0, înseamnă că subșirul nu a fost găsit iar dacă rezultatul este <>0, subșirul a fost găsit și s-a returnat numărul octetului de început al subșirului în cadrul șirului.

```
program st11;
var a,b:string;
    i:integer;

begin
  a:='calculator';
  b:='lat';
  i:=pos(b,a);
  writeln(i)
end.
```

Procedurile care urmează se prezintă fără să realizăm programe echivalente în care acestea să lipsească.

5.1.2.3. Procedura DELETE

Această procedură are rolul de a șterge un subșir din cadrul unui șir dat. Parametrii procedurii sunt următorii:

- variabila de tip string care conține subșirul din care se face ștergerea;
- poziția de început a subșirului în cadrul șirului din care se face ștergerea;
- numărul de octeți ce vor fi șterși.

În exemplul următor variabila a conține șirul 'calculator'. După execuția procedurii DELETE, variabila a va conține șirul 'calator'.

```
program st12;
var a:string;

begin
  a:='calculator';
  delete(a,4,3);
  writeln(a)
end.
```

5.1.2.4. Procedura INSERT

Rolul acestei proceduri este de a insera un șir în cadrul unui alt șir.
Pentru execuție, sunt necesari următorii parametri:

- șirul care urmează a fi inserat sau variabila ce conține acest șir;

- variabila ce conține șirul în care urmează să se facă inserarea;
- poziția primului octet din care se va face inserarea.

În programul st13 variabila a conține șirul 'mama' și variabila b conține șirul 'riti'. După execuția procedurii INSERT variabila a va conține șirul 'maritima'.

```
program st13;
var a,b:string;

begin
  a:='mama';
  b:='riti';
  insert(b,a,3);
  writeln(a)
end.
```

După cum se știe, variabilele numerice au un mod propriu de reprezentare în memoria calculatorului (cod complementar pentru variabile de tip întreg și virgula mobilă pentru cele de tip real). În cadrul variabilelor de tip string, fiecare cifră a unui număr se reține în cadrul unui octet prin codul ei (ASCII).

Se pune problema de a converti conținutul unei variabile de tip numeric într-un șir de tip string și invers. În capitolele următoare vom vedea care este motivul pentru care sunt necesare aceste conversii.

5.1.2.5. Procedura STR

Are rolul de a converti conținutul unei variabile de tip numeric într-un șir de tip string.

Pentru apelul acestei proceduri se folosesc doi parametri:
- variabila numerică ce conține numărul ce urmează a fi convertit;
- variabila de tip string care va conține numărul convertit.

După tipul variabilei ce conține valoarea numerică ce va fi convertită (întreagă sau reală) se disting două forme de apel pentru procedura STR.

a. Variabila este de tip real.

În acest caz, la apel, variabila reală se trece sub forma v:m:n unde v este numele variabilei, m numărul total de caractere pe care le va avea șirul convertit, iar n reprezintă numărul de zecimale pe care le va conține numărul convertit.

b. Variabila este de tip întreg.

La apel, aceasta va fi trecută sub forma v:m, unde m reprezintă numărul total de caractere pe care îl va avea rezultatul.

Programul st14 exemplifică convertirea conținutului unor variabile de tip real și întreg în șiruri de caractere.

```
program st14;
var a:string;
    b:real;
    c:integer;
```

```
begin
  b:=-23.25;
  str(b:6:2,a);
  writeln(a); c:=127;
  str(c:3,a);
  writeln(a)
end.
```

5.1.2.6. Procedura VAL

Această procedură are rolul de a converti un șir într-o valoare numerică. Întrucât nu orice șir poate fi convertit (de exemplu în cazul în care conține o literă) procedura VAL va avea la apel un parametru în plus și anume o variabilă de tip întreg ce va conține după apel un număr. Dacă acest număr este diferit de 0, înseamnă că operația de conversie nu a reușit.

În concluzie, procedura va utiliza la apel trei parametri:

- șirul (sau variabila de tip string ce conține șirul) care urmează a fi convertit;
- variabila de tip întreg sau real ce va conține rezultatul conversiei;
- o variabilă de tip întreg ce va conține, după execuția procedurii, numărul care ne spune dacă conversia a reușit sau nu.

În programul următor se convertesc două variabile (una de tip întreg și alta de tip real) în șiruri de caractere.

```
program st15;
var a:string;
    b:real;
    c,c_er:integer;

begin
  a:='-12.25';
  val(a,b,c_er);
  writeln(b:6:2,' ',c_er);
  a:='123';
  val(a,c,c_er);
  writeln(c,' ',c_er);
  a:='1a3';
  val(a,c,c_er);
  writeln(c,' ',c_er)
end.
```

5.2. Tipul înregistrare (RECORD)

5.2.1. Tipul înregistrare fixă

În prezentarea acestui tip, vom porni de la un exemplu practic. O firmă productivă își păstrează materialele, care urmează să intre în prelucrare, în n magazii. Se cere să ținem o evidență pe calculator a acestor materiale.

Informațiile ce trebuie reținute pentru fiecare material sunt următoarele:

- codul materialului (număr întreg);
- magazia în care se găsește (număr întreg cuprins între 1 și n);
- denumirea materialului (30 de octeți, deci STRING[30]);
- unitatea de măsură (bucăți, kilograme, litri, STRING[3]);
- cantitatea (număr real);
- preț unitar (număr real).

Dacă un material se găsește în mai multe magazii, se înregistrează de mai multe ori.

Observăm că informațiile care apar perntru fiecare material se reține în variabile de natură diferită (STRING, INTEGER, REAL). Se pune problema ca toate aceste informații să se regăsească într-o singură variabilă de un anumit tip. Tipul de variabilă care poate strânge toate aceste informații la un loc este **tipul înregistrare**. Informațiile asupra unui material existent într-o magazie constituie o înregistrare. Aceasta este alcătuită din mai multe **câmpuri** (pentru exemplul nostru, acestea sunt codul materialului, denumirea lui etc.).

Să analizăm programul de mai jos care citește și afișează informațiile referitoare la un material.

```
program rec1;
type material=record
                  cod,mag:integer;
                  den:string[30];
                  um:string[3];
                  cant,pu:real
                end;
var m:material;

begin
  write ('cod '); readln(m.cod);
  write ('magazie '); readln(m.mag);
  write ('denumire '); readln(m.den);
  write ('unitate masura ' ); readln(m.um);
  write ('cantitate  '); readln(m.cant);
  write ('pret unitar '); readln(m.pu);
  writeln ('informatiile citite ');
  writeln ('cod ',m.cod);
  writeln('magazie ',m.mag);
  writeln('denumire ',m.den);
  writeln('unitate de masura ',m.um);
  writeln('cantitate ',m.cant:5:2);
  writeln('pret unitar ',m.pu:5:2)
end.
```

Tipul material este un tip înregistrare. Descrierea unui astfel de tip începe prin cuvântul cheie **RECORD** și se termină prin **END**. Între aceste două cuvinte cheie sunt trecute toate câmpurile conținute cu tipurile lor. Partea cuprinsă între RECORD și END se numește **parte fixă**. Figurile de mai jos prezintă descrierea unui tip înregistrare cu ajutorul diagramelor de sintaxă.

Variabila m este de tip MATERIAL (deci reține toate informațiile referitoare la un

Figura 5.2.1.1.

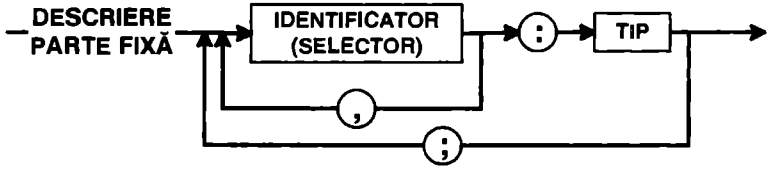


Figura 5.2.1.2.

material). Adresarea unui câmp al acestei variabile se face prin numele ei, urmat de punct și numele câmpului. Astfel, pentru a adresa câmpul den se scrie m.den.

Acest mod de adresare deranjează prin faptul că se pierde timp la scrierea programelor (de fiecare dată scriem numele variabilei). Limbajul pune la dispoziție o instrucțiune numită **WITH** care are rolul de a ușura adresarea câmpurilor. Rescriem programul anterior utilizând această instrucțiune.

```
program rec2;
type material=record
                 cod,mag:integer;
                 den:string[30];
                 um:string[3];
                 cant,pu:real
               end;
var m:material;

begin
  with m do
  begin
    write ('cod '); readln(cod);
    write ('magazie '); readln(mag);
    write ('denumire '); readln(den);
    write ('unitate masura ' ); readln(um);
    write ('cantitate  '); readln(cant);
    write ('pret unitar '); readln(pu);
    writeln ('informatiile citite ');
    writeln ('cod ',cod);
    writeln('magazie ',mag);
    writeln('denumire ',den);
    writeln('unitate de masura ',um);
    writeln('cantitate ',cant:5:2);
    writeln('pret unitar ',pu:5:2)
  end
end.
```

Forma generală a instrucțiunii WITH este:

WITH nume variabilă de tip înregistrare DO instrucțiune.

Rolul ei este de a permite (în cadrul instrucțiunii subordonate ei) adresarea unui câmp fără a preciza numele variabilei. Evident, de cele mai multe ori instrucțiunea subordonată instrucțiunii WITH este o instrucțiune compusă.
În figura de mai jos se prezintă forma generală a instrucțiunii cu ajutorul diagramelor de sintaxă.

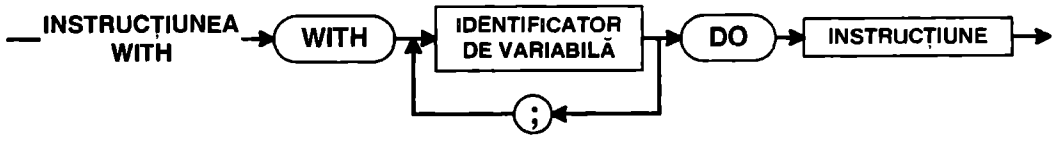


Figura 5.2.1.3.

Aplicația propusă la începutul acestui paragraf presupune lucrul cu mai multe înregistrări, dar programele prezentate până aici lucrează cu una singură. În cele ce urmează, se prezintă o modalitate de lucru cu mai multe înregistrări.

Se cere să se scrie un program care să citească n înregistrări de tip material și să calculeze valorile totale pe magazii. De asemenea se va tipări valoarea tuturor materialelor deținute de firmă.

O posibilitate de a reține cele n înregistrări este dată de declararea unui vector (structura de tip ARRAY) ale cărui componente sunt înregistrări.
Declarația acestui tip este:

type vector = array[1..100] of material.

Cum putem adresa un câmp al unei componente? Pentru aceasta, se precizează componenta urmată de punct, urmată de numele câmpului.

Exemplu: câmpul den al componentei 3 se adresează prin v[3].den, unde v este o variabilă de tip vector. Și în acest caz se poate folosi instrucțiunea WITH sub forma WITH v[i] do instrucțiune, evitând astfel o adresare greoaie. În program folosim această instrucțiune atunci când scriem secvența citirii celor n înregistrări. Pentru calculul totalurilor pe magazii, s-a reținut un alt vector, cu componente reale, numit vectot. Acest vector are un număr de componente egal cu numărul de magazii pe care le deține firma (număr pe care programul îl citește). Pentru fiecare înregistrare de tip material se calculează valoarea materialului ca produs între cantitate și preț unitar. Această valoare se adaugă totalului pe magazia căreia îi aparține materialul (câmpul mag al unei înregistrări). În final, se calculează valoarea totală a materialelor reținute de firmă prin însumarea totalurilor pe magazii.

```
program rec3;
type material=record
                  cod,mag:integer;
                  den:string[30];
                  um:string[3];
                  cant,pu:real
                end;
vector=array[1..100] of material;
vectot=array[1..10] of real;
```

```
var
  v:vector;
  n,i,nr_mag:integer;
  tot:vectot;
  total:real;

begin
  write('n=');
  readln(n);
  write('numar de magazii=');
  readln(nr_mag);
  for i:=1 to nr_mag do tot[i]:=0;
  for i:=1 to n do
    with v[i] do
    begin
      writeln('Înregistrarea ',i);
      write ('cod '); readln(cod);
      write ('magazie '); readln(mag);
      write ('denumire '); readln(den);
      write ('unitate masura ' ); readln(um);
      write ('cantitate  '); readln(cant);
      write ('pret unitar '); readln(pu);
    end;
  for i:=1 to n do
    tot[v[i].mag]:=tot[v[i].mag]+v[i].cant*v[i].pu;
  total:=0;
  for i:=1 to nr_mag do
  begin
    writeln('total magazia ',i, ' ',tot[i]:4:2);
    total:=total+tot[i]
  end;
  writeln('total general ',total:5:2)
end.
```

Observații:

- Programul prezentat este unul didactic, motiv pentru care secvența de citire se face separat (pentru a arăta cum se citește un vector de înregistrări). Se putea evita reținerea unui vector pentru această problemă. Rescrieți programul ținând cont de această observație. De asemenea, multe informații sunt inutile în program (de exemplu denumirea fiecărui material se citește pentru fiecare înregistrare).
- În practică, pentru astfel de aplicații se folosește un alt tip de înregistrare numit **FISIER** care permite memorarea informațiilor pe suport magnetic. Ce ne facem dacă la fiecare prelucrare de materiale trebuie să le introducem pe toate? Dar în construcția tipului fișier se folosește tipul înregistrare prezentat aici.

Programul care urmează utilizează din plin structura de tip vector de înregistrări. Se citesc n înregistrări de tip material. Acestea se sortează alfabetic. În final, înregistrările sortate sunt tipărite (pentru fiecare înregistrare se tipărește denumirea materialului și cantitatea). Metoda de sortare a fost prezentată pe cazul unui vector cu numere întregi (se numește metoda bulelor).

```
program rec4;
type material=record
```

```
                cod,mag:integer;
                den:string[30];
                um:string[3];
                cant,pu:real
              end;
vector=array[1..100] of material;
var v:vector;
    n,i:integer;
    inversari:boolean;
    m:material;

begin
  write('n='); readln(n);
  for i:=1 to n do
  with v[i] do
  begin
    writeln('Înregistrarea ',i);
    write ('cod '); readln(cod);
    write ('magazie '); readln(mag);
    write ('denumire '); readln(den);
    write ('unitate masura ' ); readln(um);
    write ('cantitate  '); readln(cant);
    write ('pret unitar '); readln(pu);
  end;
  repeat
    inversari:=false;
    for i:=1 to n-1 do
      if v[i].den>v[i+1].den then
      begin
        m:=v[i];
        v[i]:=v[i+1];
        v[i+1]:=m;
        inversari:=true
      end
  until not inversari;
  for i:=1 to n do
    with v[i] do
    begin
      writeln('Înregistrarea ', i); writeln(den);
      writeln(cant:4:2);
    end
end.
```

Sunt situații în care unul din câmpurile unei structuri de tip înregistrare este tot de tip înregistrare. De exemplu, în aplicația noastră, pentru fiecare material trebuie să se rețină și data fabricației. Aceasta este compusă din zi, lună și an deci are o structură de tip înregistrare.

Limbajul permite ca o structură de tip RECORD să conțină o altă structură de tip RECORD.

Analizați programul următor care citește o singură înregistrare din noul tip de structură. Pentru a adresa câmpul an al înregistrării m ar trebui să folosim adresarea m.dataf.an, adresare extrem de greoaie. Din acest motiv, în program, se folosește de două ori instrucțiunea WITH.

```
program rec5;
type material=record
                  cod,mag:integer;
                  den:string[30];
                  um:string[3];
                  cant,pu:real;
                  dataf:record
                          zi:1..31;
                          luna:1..12;
                          an:integer
                        end
                end;
var m:material;

begin
  with m do
  begin
    write ('cod '); readln(cod);
    write ('magazie '); readln(mag);
    write ('denumire '); readln(den);
    write ('unitate masura ' ); readln(um);
    write ('cantitate  '); readln(cant);
    write ('pret unitar '); readln(pu);
    with dataf do
    begin
      write('ziua fabricatiei '); readln(zi);
      write('luna fabricatiei '); readln(luna);
      write('anul fabricatiei '); readln(an)
    end
  end
end.
```

5.2.2. Tipul înregistrare cu variante

Nu toate înregistrările au o structură fixă (același număr de câmpuri) așa cum au fost cele prezentate ca exemplu în paragraful anterior. Sunt cazuri când un tip înregistrare are o parte fixă urmată de o parte variabilă.

Să presupunem că ne interesează o situație referitoare la studiile unei persoane. O astfel de înregistrare are o parte fixă dată de câmpurile care reține numele, vârsta și tipul de studii. O persoană poate să nu aibă studii, caz în care nu mai este necesar să avem alte informații, poate să fi făcut câteva clase de școală generală (ne-ar putea interesa câte clase), să fi terminat liceul (caz în care dorim să știm anul terminării și orașul) sau să aibă studii superioare (și atunci ne interesează numele facultății și numărul de ani de studiu în cadrul facultății respective). În concluzie, în funcție de tipul de studii, înregistrarea arată altfel.

Limbajul permite ca înregistrările să aibă o structură variabilă. Cum se realizează aceasta?

În primul rând trebuie reținut că partea variabilă este plasată în cadrul înregistrării după partea fixă. O parte variabilă se dezvoltă în cadrul înregistrării după valorile pe care le ia un câmp situat în cadrul părții fixe.

În programul care urmează (ce exemplifică descrierea unui tip de înregistrare variabilă) selecția părții variabile se face în funcție de valorile pe care le ia câmpul STUDII. Pentru selectare, se folosește o clauză specială numită **CASE**. Câmpul după care se face selecția apare descris în această clauză.

Câmpul selector trebuie să fie de tip ordinal cu un număr finit de elemente (tipul INTEGER nu este permis). În funcție de valorile pe care le poate lua câmpul selector se va dezvolta partea variabilă. În esență, se scriu pe rând valorile posibile ale câmpului selector. În dreptul fiecărei valori se trece partea pe care trebuie să o conțină înregistrarea în acest caz. Aceasta se încadrează între paranteze rotunde și poate fi chiar vidă. Câmpurile prezente între paranteze rotunde se scriu separate prin ';'.

Analizați programul următor (care citește o singură înregistrare):

```
program rec6;
type persoana=record
                nume:string[30];
                vârsta:byte;
                case studii:char of
                  'f':();
                  'g':(nr_cl:integer);
                  'l':(an_t:integer;
                  oras:string);
                  's':(fac:record
                            nume_f:string[20];
                            an_s:byte
                          end)
              end;
var p:persoana;

begin
  write('nume '); readln(p.nume);
  write('vârsta'); readln(p.vârsta);
  write('studii '); readln(p.studii);
  case p.studii of
    'g':  begin
            write('numar clase ');
            readln(p.nr_cl)
          end;
    'l':  begin
            write('anul terminarii liceului ');
            readln(p.an_t);
            write('orasul ');
            readln(p.oras)
          end;
    's':  begin
            write('numele facultatii ');
            readln(p.fac.nume_f);
            writeln(' ani studii facultate ');
            readln(p.fac.an_s)
          end
  end {case}
end.
```

Observații:

- Este preferabil ca citirea unei variabile de acest tip să se facă prin utilizarea instrucțiunii CASE.
- Pentru fiecare înregistrare de acest tip compilatorul rezervă un număr de octeți necesar înregistrării celei mai lungi posibile.
- Toate celelalte operații cu înregistrări de un tip variabil sunt similare cu cele prezentate pentru înregistrări de tip fix.

Cu ajutorul diagramelor de sintaxă tipul înregistrare cu variante se descrie astfel:

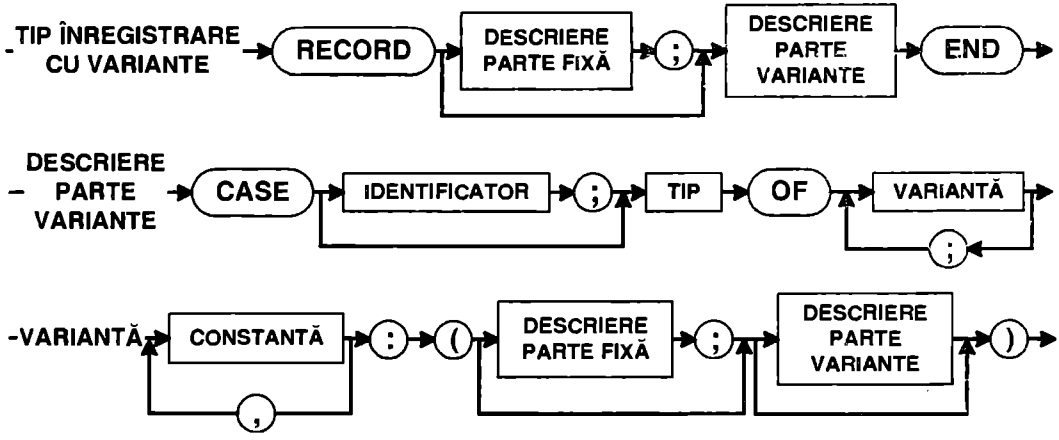


Figura 5.2.2.1.

5.3. Tipul de date mulțime

Să presupunem că se citește o variabilă de tip char. Se cere să se precizeze dacă a fost citită o literă mică a alfabetului. Cum am putea rezolva această problemă?

O posibilitate de rezolvare constă în a testa dacă am citit 'a' sau 'b' sau ... 'z'. Nu credeți că un astfel de test este cam lung? Altfel cum am putea proceda? Să reținem toate literele mici ale alfabetului într-un vector de caractere, pentru ca apoi caracterul citit să fie comparat, pe rând, cu fiecare componentă a vectorului. Nici această soluție nu strălucește prin simplitate. Limbajul Turbo Pascal oferă posibilitatea de a rezolva eficient astfel de probleme utilizând **tipul de date mulțime**.

Evident, o variabilă de un tip mulțime poate reține o mulțime de elemente. De la bun început precizăm că mulțimea nu poate avea mai mult de 256 de elemente. Oricum, este un număr suficient de mare pentru a putea rezolva cu succes o clasă largă de probleme. Ce natură au elementele care constituie mulțimile?

Aceste elemente trebuie să aparțină unui tip ordinal standard (char,boolean)

sau unui tip definit de utilizator. Tipul ordinal căruia îi aparțin elementele ce constituie mulțimile nu trebuie să conțină mai mult de 256 de caractere și se numește **tip de bază**. Ordinul elementelor ce alcătuiesc tipul de bază trebuie să fie cuprins între 0 și 255. Tipul integer nu poate fi tip de bază dar pot fi folosite subdomenii ale acestuia care au elemente între 0 și 255.

Să analizăm programul de mai jos:

```
program mul;
const a:set of char=['a'..'i','m'];
type  zile=(luni,marti,miercuri,joi);
      cifre=0..9;
      cifre_mari=10..200;
var v1:set of char;
    v2:set of zile;
    v3:set of cifre;
    v4:set of cifre_mari;

begin
  v1:=['a'..'d', 'l', 'u'..'z'];
  v1:=a;
  v2:=[];
  v2:=[luni];
  v3:=[0,1,7..9];
  v4:=[90..100];
end.
```

Variabila v1 reține o mulțime de caractere. Variabila v2 reține o mulțime de elemente aparținând tipului enumerare mai sus definit. Analizați singuri natura elementelor mulțimilor ce pot fi reținute în variabilele v3 și v4.

Declararea tipului mulțime se face așa cum arată diagrama de sintaxă de mai jos:

Figura 5.3.1.

Singura operație permisă asupra variabilelor de tip mulțime este operația de atribuire în care variabila capătă valoarea unei expresii de tip mulțime.

O expresie de tip mulțime poate conține:

- variabile de tip mulțime;
- constructori de tip mulțime;
- operatori;
- constante de tip mulțime.

5.3.1. Constructori de tip mulțime

Aceștia definesc valori de tip mulțime. Orice constructor începe prin '[' și se termină prin ']'. În esență, între paranteze drepte se trece valoarea mulțime definită de constructor. Elementele mulțimii sunt date prin enumerare.

Să analizăm constructorii prezenți în programul de mai sus:

- [] reprezintă mulțimea vidă;
- [luni] reprezintă mulțimea formată dintr-un singur element și anume elementul luni;
- [0,1,7..9] reprezintă mulțimea formată din elementele 0, 1, 7, 8 și 9.

5.3.2. Operatori

Variabilelor de tip mulțime li se pot aplica următorii operatori:

+ pentru **reuniune** (reuniunea a două mulțimi A și B este o mulțime formată din elementele mulțimii A la care se adaugă elementele mulțimii B care nu aparțin mulțimii A);

***** pentru **intersecție** (intersecția a două mulțimi A și B, reprezintă mulțimea formată din elementele comune celor două mulțimi);

− pentru **diferență** (prin diferența dintre mulțimile A și B se înțelege mulțimea formată din elementele mulțimii A care nu sunt și elemente ale mulțimii B_).

Mai avem la dispoziție și operatori relaționali:

in pentru **apartenență**;

< > pentru a testa dacă două mulțimi sunt **diferite** (nu conțin aceleași elemente);

<= pentru a testa dacă o mulțime este **inclusă** în alta;

>= pentru a testa dacă o mulțime **include** altă mulțime.

5.3.3. Constante

În primul program al acestui paragraf se definește o constantă de tip mulțime. Aceasta se declară prin definirea tipului mulțime, urmat de semnul '=', după care se trece valoarea mulțime sub formă de constructor.

Elementele unei mulțimi nu pot fi scrise și nici citite în mod direct. Pentru a putea realiza aceste operații se recurge la mici artificii.

Cum citim o variabilă mulțime?

Pentru a realiza acest lucru se procedează astfel:

- variabila mulțime se inițializează (acesteia i se atribuie mulțimea vidă);
- se citește un element (într-o variabilă având același tip ca tipul de bază al mulțimii);
- elementul citit este privit ca mulțime cu un singur element, iar această mulțime se reunește cu mulțimea care se citește;
- se citește un alt element cu care se procedează la fel, până când au fost citite toate elementele.

Pentru a realiza tipărirea, element cu element, a unei mulțimi se procedează astfel:

- se consideră o variabilă de același tip cu tipul de bază al mulțimii;
- această variabilă este folosită (ca variabilă de ciclare) în construcția unui ciclu FOR care are ca valoare inițială prima valoare posibilă a mulțimii și ca valoare finală, ultima valoare posibilă a mulțimii.
- la fiecare pas se testează dacă valoarea de ciclare aparține sau nu mulțimii (în caz afirmativ aceasta se tipărește).

Programul care urmează citește două mulțimi A și B după care tipărește reuniunea, intersecția și diferența celor două mulțimi. Elementele celor două mulțimi sunt numere naturale cuprinse între 0 și 255.

```
program mul1;
type m_int=set of 0..255;
var a,b,c:m_int;
    m,n,i,nr:integer;

begin
  write('card(a)=');
  readln(m);
  a:=[];
  for i:=1 to m do
  begin
    write('elementul ',i,' ');
    readln(nr);
    a:=a+[nr]
  end;
  writeln('multimea A');
  for i:=0 to 255 do
  if i in a then writeln(i);
  readln;
  write('card(b)=');
  readln(n);
  b:=[];
  for i:=1 to n do
  begin
    write('elementul ', i,' ');
    readln(nr);
    b:=b+[nr];
  end;
  writeln('multimea B');
  for i:=1 to 255 do
  if i in b then writeln(i);
  readln;
  c:=a+b;
  writeln ('multimea reuniune');
  for i:=1 to 255 do
  if i in c then writeln(i);
  readln;
  c:=a*b;
  writeln('multimea intersectie');
  for i:=1 to 255 do
  if i in c then writeln(i);
  readln;
  c:=a-b;
  writeln('A-B');
  for i:=1 to 255 do
  if i in c then writeln(i);
  readln
end.
```

Observații:

- Nu orice tip de bază al unei mulțimi poate fi citit sau tipărit (*exemplu*: tipul

enumerare). Pentru cazuri în care avem de citit sau scris mulțimi cu aceste tipuri de bază se recurge la următoarele artificii:

Citirea. Se citește, de exemplu, o variabilă de tip string și în funcție de valoarea citită se reunește mulțimea care trebuie citită cu elementul corespunzător cuvântului citit.

Scrierea. Se tipărește un cuvânt corespunzător elementului care trebuie tipărit.

- De la matematică, știm că o mulțime are elemente distincte. Orice variabilă de tip mulțime va avea numai elemente distincte. De fapt, nici nu poate fi altfel. O mulțime se construiește utilizând numai operații cu mulțimi care au ca rezultat mulțimi. Să presupunem că atunci când se rulează programul MUL1, la citirea mulțimii A se introduce de două ori elementul 2. La fiecare introducere, mulțimea A este reunită cu acel element, deci se va reuni mulțimea A de două ori cu elementul 2. Rezultă că mulțimea nu va conține decât o singură dată elementul 2 (în caz contrar nu se efectuează corect operația de reuniune). Acest fapt nu este lipsit de importanță. Să presupunem că dorim să știm câte litere distincte are un cuvânt. Este suficient să construim mulțimea literelor lui. Cum o literă nu poate figura de două ori într-o mulțime de litere, numărul de elemente al mulțimii va fi egal cu numărul de litere distincte ale cuvântului.

5.3.4. Constante de tip mulțime

În primul program propus în acest paragraf am definit o constantă de tip mulțime. Declararea unei constante se face prin declararea tipului ei, urmat de semnul egal, după care se scrie constructorul mulțime.

Deși tipul mulțime este folosit rar în scrierea programelor, sunt cazuri când este de maxim folos. De exemplu, dacă dorim să știm dacă o variabilă de tip string conține numai litere (test alfabetic), se verifică apartenența fiecărui caracter la mulțimea literelor. Gândiți-vă cum ar arăta un program care face un astfel de test fără a utiliza tipul mulțime.

Programul MUL2 numără vocalele dintr-un text citit.

```
program mul2;
const v:set of char=['a','e','i','o','u','A','E','I','O','U'];
var a:string;
    i,s:integer;

begin
  write(' Dati textul ');
  readln(a);
  s:=0;
  for i:=1 to length(a) do
    if a[i] in v then s:=s+1;
  writeln('avem ',s,' vocale' );
end.
```

5.4. Aplicații la capitolul 5

Aplicații asupra tipurilor de date structurate

5.4.1. Tipul ARRAY

1. Să se citească și să se tipărească un vector cu componente de tip CHAR.

2. Să se citească și să se tipărească o matrice cu componente de tip REAL.

3. Se citește un număr natural N. Să facem conversia acestui număr în baza 2. Cifrele de 1 sau 0 se vor reține într-un vector. Să se tipărească vectorul (atenție, nu se vor tipări cifrele zero nesemnificative).

4. Aceeași problemă ca precedenta pentru conversia în baza 16.

5. Se citește un număr întreg N (atenție, poate fi și negativ). Să se convertească în cod complementar pe 16 poziții binare. Cifrele de 1 sau 0 se vor reține într-un vector. Să se tipărească rezultatul.

6. Se citește un număr natural N (citirea se va face într-o variabilă de tip INTEGER). Să se scrie acest număr într-un vector.

Exemplu: Citim 1231. Vom avea V[1]=1, V[2]=2, V[3]=3, V[4]=1.

7. Se citește un vector cu n componente numere naturale. Să se tipărească cel mai mare divizor comun al celor n numere.

8. Se citește un vector cu componente numere naturale. Să se tipărească numărul cifrelor de 0 cu care se termină numărul format din produsul celor n componente.

9. Să se calculeze 2 la puterea 100;

10. Un număr natural se reține într-un vector astfel încât fiecare componentă a vectorului conține o cifră a numărului. *Exemplu*: dacă vectorul V are 5 componente iar numărul este 128, V[1]=0, V[2]=0, V[3]=1, V[4]=2, V[5]=8. Să se înmulțească numărul cu un alt număr natural format dintr-o singură cifră.

11. Aceeași problemă ca precedenta, însă deînmulțitul are mai multe cifre și este reprezentat într-un vector.

12. Se citesc într-un vector n numere naturale: a1, a2, ..., an. Să se calculeze:

$S1 = a1 + a2 + \ldots + an;$
$S2 = a1 * a2 + \ldots + a1 * an + a2 * a3 + \ldots + a2 * an + \ldots + an-1 * an$
.
.
$Sn = a1 * a2 * \ldots * an.$

13. Se citește un vector cu n componente numere naturale distincte. Acestea alcătuiesc o mulțime de numere naturale. Se cere să se tipărească

toate submulțimile acestei mulțimi.

14. Se citesc doi vectori cu componente numere naturale. Fiecare vector are elementele sortate crescător. Se cere să se construiască un al treilea vector care conține componentele celor doi în ordine crescătoare. Aceasta este problema interclasării și are o mare importanță practică.

15. Se consideră un vector cu componente numere naturale distincte și sortate crescător. Se citește un număr natural M. Se cere să se precizeze dacă numărul citit se găsește sau nu între componentele vectorului considerat, iar în caz afirmativ să se tipărească indicele componentei care îl conține (căutare binară).

16. Se consideră F un polinom de gradul n, unde cei n+1 coeficienți sunt reținuți într-un vector. Se citește o valoare reală a. Să se tipărească F(a).

17. Se dau două polinoame ai căror coeficienți sunt reținuți în doi vectori. Să se calculeze coeficienții produsului celor două polinoame.

18. Se consideră un vector care conține n numere naturale. Să se tipărească o submulțime a acestora a cărei sumă se divide cu n.

19. Tipăriți suma elementelor aflate deasupra diagonalei principale a unei matrice pătratice.

20. Tipăriți suma elementelor aflate sub diagonala principală a unei matrice pătratice.

21. Tipăriți suma elementelor aflate sub diagonala secundară a unei matrice pătratice.

22. Interschimbați coloanele unei matrice cu m linii și n coloane astfel încât în linia k elementele să fie în ordine strict crescătoare.

23. Interschimbați linii și coloane ale unei matrice cu n linii și n coloane astfel încât elementele de pe diagonala principală să fie sortate crescător.

24. Determinați elementele șa ale unei matrice cu n linii și m coloane (elemente minime pe linie și maxime pe coloană sau maxime pe linie și minime pe coloană).

25. Tipăriți sumele elementelor aflate pe pătrate concentrice ale unei matrice pătratice.

Indicații :

8. Nu putem efectua produsul celor n numere pentru ca, apoi, să vedem cu câte cifre de 0 se termină, pentru că într-un astfel de caz numărul obținut poate fi foarte mare (nu poate fi reținut într-o variabilă de tip integer sau longint). Soluția este următoarea:

- se determină k1, exponentul maxim al cifrei 2 astfel încât 2 la puterea k1 divide pe V[1];
- ...
- se determină kn, exponentul maxim al cifrei 2, astfel încât 2 la puterea kn divide V[n];
- se calculează a = k1 + k2 + ... + kn (am determinat puterea maximă a lui doi

astfel încât 2 la puterea a divide produsul considerat);
- se determină p1, exponentul maxim al cifrei 5, astfel încât 5 la puterea kn divide pe v[1];
- se determină pn, exponentul maxim al cifrei 5, astfel încât 5 la puterea pn divide V[n];
- se calculează b = p1 + p2 + ... + pn puterea maximă a lui 5 astfel încât 5 la puterea b divide produsul considerat.

Numărul de cifre 0 cu care se termină produsul va fi minimul între a și b (de ce?).

9. Dacă am calcula produsul pe căi clasice, numărul obținut este atât de mare încât nu se poate reprezenta în memorie nici chiar prin utilzarea tipului longint. Pentru aceasta, suntem nevoiți să simulăm înmulțirea cu 2 reținând cifrele numărului într-un vector.

Astfel, se pornește de la un vector ale cărui componente au valorile:
0 0 0 0 0... 0 0 0 1 (numărul 1);
- se înmulțește fiecare componentă a vectorului cu 2:

0 0 0 0 0... 0 0 0 2;
- se înmulțește fiecare componentă cu 2

0 0 0 0 0... 0 0 0 4
0 0 0 0 0... 0 0 0 8
0 0 0 0 0... 0 0 0 16;
- după fiecare înmulțire cu 2 se parcurge vectorul în sens invers și pentru fiecare componentă (n) cu un conținut mai mare decât 9 se scade 10 și se adună 1 la componenta n−1:

0 0 0 0 0... 0 0 0 1 6
- se va obține în continuare:

0 0 0 0 0... 0 0 0 **2 12**
0 0 0 0 0... 0 0 0 3 2
0 0 0 0 0... 0 0 0 6 4

Estimați numărul de cifre pe care îl va avea numărul 2 la puterea 100 astfel încât să reținem tipul vector cu atâtea componente câte sunt necesare.

12. Sumele considerate sunt coeficienții polinomului:
$P = (x + a_1)(x + a_2)...(x + a_n)$.

Astfel:

$(x + a_1)(x + a_2) = x^2 + (a_1 + a_2)x + a_1a_2$

$(x + a_1)(x + a_2)(x + a_3) = x^3 + (a_1 + a_2 + a_3)x^2 + (a_1a_2 + a_1a_3 + a_2a_3)x + a_1a_2a_3$

$(x + a_1)(x + a_2)(x + a_3)...(x + a_n) =$

$= x^n + (a_1 + ... + a_n)x^{n-1} + (a_1a_2 + ... + a_{n-1}a_n)x^{n-2} + ... + a_1a_2...a_n$

$x + a_1$ este un polinom ai cărui coeficienți se reține într-un vector în felul următor: 0 0 0 0 0 0...1 a1;
- se scrie o secvență care înmulțește un polinom cu x + a;
- această secvență se folosește de n−1 ori (pentru $a = a_2$, $a = a_3$,..., $a = a_n$.

În final, se tipăresc coeficienții lui x la puterea n−1 (S1), x la puterea n−2 (S2), x la puterea 0 (Sn).

13. Există mai mulți algoritmi care rezolvă problema considerată. Aici, vom studia unul de o mare simplitate.
Fie vectorul considerat:
a1 a2 a3 a4 ... an.

Acestuia i se atașează un vector ale cărui componente pot lua numai valori 0 sau 1 (pentru început 0). Un astfel de vector se numește vector caracteristic. Astfel avem:
0 0 0 0 ... 0.
Pentru ca acest vector să reprezinte o submulțime, facem următoarea convenție: elementul conținut în componenta k a primului vector aparține sau nu submulțimii considerate după cum componenta k a vectorului caracteristic are valoarea 1 sau 0. Astfel, a genera toate submulțimile mulțimii considerate se reduce la generarea în cadrul vectorului caracteristic a tuturor combinațiilor posibile de 0 sau 1. Cum rezolvăm această ultimă problemă? Simplu, considerăm că vectorul caracteristic reține un număr în baza 2 (inițial 0, corespunzător submulțimii vide). La fiecare pas se adună 1 acestui număr (se simulează adunarea cu 1 a numărului conținut în vectorul caracteristic) și se tipărește submulțimea obținută (se tipăresc acele componente ale primului vector cărora le corespunde 1 în cadrul vectorului caracteristic). Astfel avem:

0 0 0 0 0 0 ..... 0 (submulțimea vidă);
0 0 0 0 0 0 ..... 1 {an};
0 0 0 0 0 0 ..... 0 1 0 {an–1};
0 0 0 0 0 0 ..... 0 1 1 {an–1,an};
0 0 0 0 0 0 ... 0 1 0 0 {an–2};
0 0 0 0 0 0 ... 0 1 0 1 {an–2,an};
...
1 1 1 1 1 1 ... 1 1 1 1 {a1,a2,...an}.

Dacă ținem seama de faptul că cel mai mare număr care se poate scrie în baza 2 prin utilizarea a n componente care reținn 0 sau 1 este doi la puterea n–1, regăsim faptul că o mulțime cu n elemente are doi la puterea n submulțimi (inclusiv cea vidă), rezultat bine cunoscut de la matematică.

14. Interclasare. Să presupunem că se citesc 2 vectori A și B, primul cu m componente, al doilea cu n componente. Cei doi vectori se introduc deja sortați. O primă posibilitate de lucru ar fi să-i trecem pe amândoi în cadrul vectorului C, pe care apoi să-l sortăm. În acest fel nu beneficiem în nici un fel de faptul că cei doi vectori sunt deja sortați (se consumă mult timp inutil). Prezentarea algoritmului se va face pe un exemplu:

A 1 3 4 5 8 (m=5);
B 2 4 5 6 7 9 10 (n=7);
Variabila i va reține indicele componentei la care s-a ajuns în vectorul A. Variabila j reține indicele la care s-a ajuns în parcurgerea vectorului B. Variabila k reține indicele elementului care urmează a fi scris în cadrul vectorului C.
Inițial avem i=1, j=1, k=1.

- Se compară A[1] cu B[1]. Este mai mic A[1], deci C[1] va fi 1, i va fi 2, k=2.
- Se compară A[2] cu B[1]. Este mai mic B[1], deci C[2]=2, k=3, j=2.
- Se compară A[2] cu B[2], este mai mic A[2], C[3]=3, k=4, i=3.

Procedeul se repetă atât timp cât și i este mai mic sau egal cu m și j este mai mic sau egal cu n. Este clar că, la un moment dat, unul din vectori a fost analizat complet (componentele sale au fost trecute în vectorul C). Componentele rămase din celălalt vector se copiază și ele în vectorul C.

```
program interclas;
type vector=array[1..100] of integer;
var a,b,c:vector;
    i,j,k,m,n:integer;

begin
  write('m='); readln(m);
  write('n='); readln(n);
  for i:=1 to m do
  begin
    write('a[',i,']=');
    readln(a[i]);
  end;
  for j:=1 to n do
  begin
    write('b[',j,']=');
    readln(b[j])
  end;
  i:=1; j:=1; k:=1;
  while(i<=m) and (j<=n) do
  begin
    if a[i]<b[j] then
    begin
      c[k]:=a[i];
      i:=i+1
    end
    else
    begin
      c[k]:=b[j];
      j:=j+1
    end;
    k:=k+1
  end;
  if i<=m then
    for j:=i to m do
    begin
      c[k]:=a[j];
      k:=k+1
    end
  else
    for i:=j to n do
    begin
      c[k]:=b[i];
      k:=k+1
    end;
  for i :=1 to k-1 do writeln(c[i]);
end.
```

15. Căutare binară. În cazul în care vectorul nu era sortat, valoarea m ar fi fost căutată prin comparație cu conținutul celorlalte componente, începând cu prima, până când era găsită (caz în care se tipărea indicele), sau până când am fi parcurs în întregime vectorul fără să o găsim. Faptul că vectorul este deja sortat ne ajută să rezolvăm problema mai rapid. Care este principiul de căutare? În esență, acesta constă în următoarele:

- avem două limite în care facem căutarea: limita inferioară (li) și limita

superioară (ls);
- inițial li = 1 și ls = n;
- se compară valoarea lui m cu valoarea componentei din mijloc (mai corect, de indice (li + ls) DIV 2) și apar trei posibilități:
  - egalitate, caz în care se tipărește indicele și algoritmul se termină;
  - componenta din mijloc este mai mică decât valoarea căutată (aceasta înseamnă că avem șanse să găsim valoarea căutată de la componenta cu indicele imediat superior componentei care a fost comparată și cea de indice ls), caz în care li va lua ca valoare indicele componentei care a fost comparată, la care se adună 1, iar limita superioară rămâne nemodificată;
  - componenta din mijloc este mai mare decât valoarea căutată și, din motive asemănătoare celor expuse anterior, limita superioară devine indicele componentei care a fost comparată, din care se scade 1, iar limita inferioară rămâne nemodificată;
    - în ultimele două cazuri algoritmul se reia până când limita inferioară devine mai mare decât limita superioară, sau până când componenta a fost găsită.

Observație. După efectuarea unei comparații, numărul valorilor în care căutăm se reduce la jumătate. Câștigul de timp este foarte mare. Să ne imaginăm că avem 1000 de componente. Cu cel mult 10 comparații am rezolvat problema (2 la puterea 10 este 1024), altfel ar fi trebuit efectuate (în cazul cel mai nefavorabil) 1000 de comparații.

```
program cbin;
type vector=array[1..100] of integer;
var v:vector;
    li,ls,lm,i,n,m:integer;
    gasit:boolean;
begin
  write('n='); readln(n);
  for i:=1 to n do
  begin
    write('v[',i,']=');
    readln(v[i])
  end;
  write('m='); readln(m);
  li:=1;
  ls:=n;
  gasit:=false;
  while (li<=ls) and not gasit do
  begin
    lm:=(li+ls) div 2;
    if v[lm]<m   then li:=lm+1
                 else  if v[lm]=m   then
                       begin
                         writeln(lm);
                         gasit:=true
                       end
                       else ls:=lm-1
  end
end.
```

16. Problema nu este dificilă. Totul este de a nu calcula la fiecare pas a la puterea k = a*a*a*...*a (de k ori). Nu uitați că la pasul anterior am calculat a la puterea k−1.

```
program polinom;
type coef=array[0..20] of real;
var c:coef;
    n,i:integer;
    a,f,p:real;

begin
  write ('n='); readln(n);
  for i:=0 to n do
  begin
    write('a',i,'=');
    readln(c[i])
  end;
  p:=1;
  write('a='); readln(a);
  f:=c[n];
  for i:=n-1 downto 0 do
  begin
    p:=p*a;
    f:=f+c[i]*p
  end;
  writeln('f(',a:3:2,')=',f:3:2)
end.
```

18. Oricât ar părea de curios, există întotdeauna o secvență de numere consecutive a căror sumă se divide la n. Să demonstrăm acest fapt. Calculăm sumele: $S1 = a1$;
$S2 = a1 + a2$;
:
$Sn = a1 + a2 + \ldots + an$.

Dacă suma Sk se divide cu n problema este rezolvată (se tipăresc a1, a2, ...,ak). Să presupunem că nici o sumă nu se divide cu n. Știm de la matematică faptul că restul împărțirii la n poate fi {0,1,...,n−1}. Cazul restului 0 a fost tratat anterior. Rezultă că nu putem avea decât un rest cuprins între 1 și n−1. Întrucât avem n sume și n−1 posibilități de rest, rezultă că există două sume care dau același rest la împărțirea la n. Fie acestea Si și Sj (i<j).

$Si = a1 + a2 + \ldots + ai$;
$Sj = a1 + a2 + \ldots + ai + (ai + 1) + \ldots + aj$;
$Sj - Si = (ai + 1) + \ldots + aj$.

Se tipăresc ai+1, ai+2,...,aj (diferența a două numere care dau același rest prin împărțirea cu n se divide la n). Utilizând cele prezentate, scrieți un program corespunzător.

19-21. Pentru identificarea elementelor care intră în calculul sumei nu trebuie parcursă în întregime matricea. Găsiți o relație, între indicii acestor elemente, care să permită o identificare directă a lor.

23. Sortarea se realizează prin comparația și eventual inversarea elementelor de forma A[i,i] și A[i+1,i+1]. Dacă este cazul, inversarea se realizează prin interschimbarea liniilor i și i+1 și interschimbarea coloanelor i și i+1.

24. *Atenție!* Maximele sau minimele pe linii sau coloane pot să nu fie unice. Dacă vom considera numai o valoare dintre acestea, s-ar putea să pierdem soluții.

5.4.2. Tipul STRING

1. Scrieți o secvență care preia de la tastatură o variabilă numerică, în felul următor:
- se citește o variabilă de tip STRING;
- conținutul acesteia se convertește într-o variabilă de tip real;
- în cazul în care conversia nu reușește (de exemplu am introdus o literă), procedeul se reia.

2. Se citește într-o variabilă de tip string un text. Cuvintele se consideră separate prin virgulă, spațiu sau punct. Câte cuvinte are textul citit?

3. Se citește într-o variabilă de tip string un text și în alta o succesiune de caractere. De câte ori întâlnim această succesiune în cadrul textului?

4. Se citesc n cuvinte și se cere ca acestea să fie sortate alfabetic.

5. Se citește într-o variabilă de tip string un text. Două cuvinte pot fi separate printr-unul sau mai multe spații. Se cere să se elimine spațiile inutile.

5.4.3. Tipul RECORD

1. Citiți o variabilă de tip înregistrare care are următoarea structură:
- nume elev: string[30]
- data nașterii :record
  zi :string[2];
  luna :string[2];
  an :string[4];
- nota matematică, nota informatică, media celor două note: real (media se calculează, nu se citește).

Testați dacă datele au fost introduse corect. Citirea se va verifica prin tipărirea rezultatului. Programul va utiliza instrucțiunea (mai corect, directiva de compilare) WITH.

2. Citiți n înregistrări de tipul celei de mai sus și afișați-le în ordinea alfabetică.

3. Aceeași problemă numai că afișarea se va face în ordinea descrescătoare a mediilor.

4. Presupunând că înregistrările se referă la un examen de admitere, să se tipărească în ordinea descrescătoare a mediilor în limita unui număr de locuri sau până când se epuizează toate înregistrările elevilor cu medii mai mari sau

egale cu 5. În cazul în care pe ultimul loc avem mai mulți elevi cu aceeași medie, toți aceștia sunt considerați admiși. Programul va afișa numărul de locuri necesare în plus.

5. Un serviciu de evidență a populației reține pentru fiecare persoană următoarele:
- nume: string[30];
- vârsta: byte;
- apartenența politică (1 dacă persoana este înscrisă într-un partid, 0 în caz contrar). În situația apartenenței persoanei la un partid politic se reține și numele partidului (string [20]).

Scrieți un program care să citească informații despre n astfel de persoane și să tipărească structura persoanelor pe opțiuni politice.

5.4.4. Tipul mulțime

1. Se citește numele unei persoane într-o variabilă de tip string. Se cere să se testeze natura alfabetică a acestuia (să nu conțină cifre).

2. Se citesc n mulțimi de numere naturale mai mici sau egale cu 255. Se cere să se tipărească dacă aceste mulțimi sunt disjuncte sau nu. (Două multimi sunt disjuncte dacă nu au elemente comune.)

3. *Problema partiției.* Se consideră o mulțime finită A. Mulțimile A1, A2,...,Am alcătuiesc o partiție a mulțimii A dacă îndeplinesc simultan trei condiții:
- toate mulțimile sunt incluse în A;
- sunt disjuncte (oricum am intersecta două dintre ele, intersecția este vidă);
- reuniunea tuturor acestor mulțimi este mulțimea A.

Se citesc m, cele m mulțimi A1, A2,...,Am (de numere între 0 și 255) și o mulțime A. Se cere să se răspundă la întrebarea dacă mulțimile citite alcătuiesc sau nu o partiție a mulțimii A.

4. *Partiția determinată de o relație de echivalență.* Se consideră o mulțime A. O relație oarecare R între elementele acestei mulțimi este o relație de echivalență dacă respectă următoarele trei condiții:
- oricare ar fi x aparținând mulțimii A, x R x (x este echivalent cu x), proprietate numită *reflexivitate*;
- oricare ar fi x și y aparținând mulțimii A, din x R y rezultă y R x, proprietate numită *simetrie*;
- oricare ar fi x, y, z, aparținând mulțimii A, din x R y și y R z, rezultă x R z, proprietate numită *tranzitivitate*.

Exemplu. În mulțimea triunghiurilor considerăm relația de asemănare. Aceasta este o relație de echivalență deoarece:
- orice triunghi este asemenea cu el ;
- dacă un triunghi este asemenea cu un al doilea, atunci și acesta este asemenea cu el;
- dacă un triunghi este asemenea cu un altul și acesta este asemenea cu un al treilea, atunci primul triunghi este asemenea cu al treilea.

Observație. Nu toate relațiile între elementele unei mulțimi sunt relații de echivalență. *Exemplu*: în mulțimea numerelor naturale relația de divizibilitate nu este simetrică (a divide b nu implică b divide a).

O relație de echivalență determină o partiție a unei mulțimi (fiecare submulțime a partiției conține elementele echivalente între ele). Cum orice element este cel puțin echivalent cu el însuși rezultă că acesta aparține unei mulțimi a partiției. Dacă două submulțimi ar avea cel puțin un element comun rezultă că toate elementele din prima submulțime și toate elementele din cea de-a doua submulțime sunt echivalente între ele (tranzitivitatea prin utilizarea elementului comun), deci cele două mulțimi coincid.

Se citește o mulțime de numere între 0 și 255 prin citirea a n perechi (x y) de numere de acest tip. Printr-o astfel de pereche se înțelege că x este echivalent cu y. Se cere să se determine partiția generată de relația de echivalență considerată pe mulțime.

Exemplu: citim 1 2
4 5
2 3
6 7
7 1

Se obține partiția: {1, 2, 3, 6, 7} {4, 5} a mulțimii {1, 2,...,7}.

Vom prezenta un algoritm neperformant din punct de vedere al timpului de calcul, însă accesibil în acest moment. În ce constă acest algoritm?

Toate elementele mulțimii considerate sunt citite sub forma (x,y). Printr-o astfel de relație se înțelege că x este echivalent cu y. Numărul de perechi care se citesc este cunoscut de la început (n). O variabilă (m) reține mulțimile care se formează (este un vector de mulțimi). O altă variabilă (nr_m) reține indicele maxim al mulțimilor create (inițial are valoarea 0). La citirea unei perechi (x,y) se ajunge într-una din cele trei situații care vor fi analizate în continuare.

- Nici x, nici y nu aparțin unei mulțimi a partiției. În acest caz se creează o nouă mulțime care va include cele două elemente considerate (nr_m crește cu o unitate).
- Numai x sau numai y se găsește într-una din mulțimile partiției. În acest caz, celălalt element este adăugat acelei mulțimi (de fapt mulțimea în care a fost găsit un element se reunește cu mulțimea formată din cele două elemente și după regulile incluziunii elementul deja conținut nu se mai adaugă).
- Elementul x se găsește într-o mulțime și elementul y în alta. În acest caz cele două mulțimi se reunesc în cadrul mulțimii care are cel mai mic indice în vector. În cazul în care cealaltă mulțime era cu indicele cel mai mare nr_m scade cu o unitate, în caz contrar această mulțime capătă ca valoare mulțimea vidă.

Pentru a testa apartenența elementelor x și y mulțimilor considerate, se folosesc două variabile GASIT și INDICE_M. De câte ori este găsit un element într-o mulțime, conținutul variabilei GASIT crește cu o unitate. În final, aceasta poate avea trei valori: 0, 1 sau 2. Variabila INDICE_M (vector cu componente de tip byte) reține de fiecare dată indicele mulțimii în care se găsește x sau y. În cazul în care GASIT are valoarea 0 conținutul variabilei INDICE_M nu interesează, dacă GASIT are valoarea 1 interesează numai conținutul primei componente a variabilei INDICE_M, iar dacă GASIT are valoarea 2 ambele

componente prezintă interes (reţin indicii celor două mulţimi).

La sfârşit listăm mulţimile partiţiei obţinute.

```
program echiv;
type  multime=set of byte;
      vector_multimi=array[1..50] of multime;
var n,i,j,x,y,nr_m:byte;
    gasit:0..2;
    m:vector_multimi;
    indice_m:array[1..2] of byte;

begin
  write('n='); readln(n);
  nr_m:=0;
  for i:=1 to n do
  begin
    write('(x,y)='); readln(x,y);
    gasit:=0;
    for j:=1 to nr_m do
      if (x in m[j]) or (y in m[j]) then
      begin
        gasit:=gasit+1;
        indice_m[gasit]:=j;
      end;
    case gasit of
      0:  begin
            nr_m:=nr_m+1;
            m[nr_m]:=[x,y];
          end;
      1:  m[indice_m[1]]:=m[indice_m[1]]+[x,y];
      2:  begin
            m[indice_m[1]]:=m[indice_m[1]]+m[indice_m[2]];
            if indice_m[2]=nr_m  then nr_m:=nr_m-1
                                 else m[indice_m[2]]:=[]
          end;
    end {case}
  end;
  writeln (' Partitia obtinuta '); writeln;
  for i:=1 to nr_m do
    if m[i]<>[] then
    begin
      writeln ('multime ');
      for j:=0 to 255 do
        if j in m[i] then write (j, ' ');
      writeln
    end;
end.
```

Capitolul 6

Subprograme

6.1. Conceptul de subprogram

Blocul este unitatea de bază a oricărui program Pascal. El este format din două părți:
- o parte de declarații (constante, tipuri, variabile, proceduri și funcții), opțională;
- o instrucțiune compusă, obligatorie.

Din cele spuse rezultă că orice program Pascal este un bloc.

În practica programării, apar următoarele situații care necesită o rezolvare:
- o anumită secvență dintr-un program se repetă;
- există multe programe care au nevoie de un anumit calcul (fie el oricât de complex).

Apare ideea că acea secvență (calculul) să fie scrisă o dată și să fie folosită de câte ori este necesar. În acest mod, am ajuns la noțiunea de **subprogram**.

Subprogramul reprezintă o parte dintr-un program, identificabilă prin nume, care se poate lansa în execuție ori de câte ori este cazul.

Exemplu: Multe programe (din domeniile cele mai diverse), pentru a putea ajunge la rezultat, trebuie să rezolve un sistem de ecuații liniare. În acest caz, se va scrie un subprogram care rezolvă un astfel de sistem.

Care trebuie să fie structura unui astfel de subprogram?
Pentru a răspunde la această întrebare, trebuie să ne gândim ce trebuie să conțină el. În primul rând, este necesară o secvență de calcul propriu-zis (care poate fi efectuată într-o instrucțiune compusă).

Pentru a fi general, subprogramul nostru trebuie să lucreze cu variabile proprii și pentru acestea trebuie declarat tipul. Este posibil ca și acest subprogram să necesite la rândul lui alte subprograme și acestea să fie declarate la rândul lor.

Pe scurt, un subprogram trebuie să fie la rândul lui un bloc.
În Pascal, subprogramele sunt de două tipuri: **proceduri** și **funcții**.

6.2. Domeniul de vizibilitate al identificatorilor

În paragraful anterior s-a arătat că un program (care este un bloc) poate apela unul sau mai multe subprograme care la rândul lor sunt blocuri. Orice subprogram poate utiliza alte subprograme ș.a.m.d. În practică, aceste

descompuneri sunt de o mare diversitate.

Să presupunem că la nivelul unui bloc am declarat o variabilă.

Exemplu a: integer. Se pune următoarea întrebare: în care din blocurile întregului program este recunoscută variabila a ? Prin faptul că ea este recunoscută, înțelegem că ne putem referi la ea (de exemplu, să efectuăm atribuirea a:=a+1) fără să apară mesajul de identificator necunoscut. Răspunsul la această întrebare este următorul: variabila este recunoscută în blocul în care am declarat-o și în toate blocurile care au fost definite în acesta. Într-un bloc care include blocul în care am declarat această variabilă, ea nu este recunoscută. Pentru blocul în care am definit-o, variabila este **locală**, pentru blocurile care au fost declarate în acesta, variabila este **globală**, iar pentru blocurile care includ blocul în care a fost declarată ea este **necunoscută**. O excepție de la acest principiu apare atunci când redeclarăm o variabilă. Ce înțelegem prin asta? La nivelul unui bloc am declarat variabila a de tip întreg. Într-un bloc subordonat acestuia (declarat în cadrul lui) declar din nou variabila a ca fiind de tip real. Dacă în acest bloc acționez asupra acestei variabile, ea este văzută ca una de tip real. Pentru a explica acest mecanism, am prezentat **domeniul de vizibilitate** al unei variabile. În realitate, mecanismul este același și pentru identificatorii de constante, tipuri, proceduri și funcții.

În concluzie, reținem următoarele: domeniul de vizibilitate al unui identificator este blocul în care acesta a fost definit și toate blocurile definite în cadrul acestuia, excepție făcând cazul unei redefiniri.

În paragraful următor vor fi date multe exemple în acest sens.

6.3. Proceduri

6.3.1. Declarare și apel

Procedura este un subprogram care se caracterizează prin faptul că rezultatele prelucrării (excepție făcând operațiile de ieșire) se găsesc depuse exclusiv în anumite variabile.

Unii autori diferențiază procedura de funcție, prin aceea că procedura poate returna mai multe rezultate.

Declarația procedurii (procedurilor) se face în blocul apelant după declarația variabilelor. În acest paragraf vom considera numai proceduri fără parametri. În acest caz, o declarație constă în a scrie cuvântul cheie **PROCEDURE** urmat de **numele procedurii.**

Cu excepția declarației, structura unei proceduri este aceeași cu structura unui program, deci aceea de bloc.

Apelul unei proceduri se face prin numele ei. Atunci când în blocul apelant a fost întâlnit un astfel de **apel** (se mai numește și **instrucțiune de apel**), se întrerupe secvența în curs, se predă controlul procedurii și după execuția acesteia se revine la instrucțiunea imediat următoare celei de apel. În

continuare, se prezintă câteva exemple de apel proceduri, exemple de un interes exclusiv didactic.

Exemplul 1

```
program pr1;
var a,b,c:integer;

  procedure suma;
  begin
    c:=a+b
  end;

begin
  write('a=');
  readln(a);
  write('b=');
  readln(b);
  suma;
  write('c=',c)
end.
```

Programul pr1 citește două numere a și b și calculează suma lor. Calculul sumei se face în procedura 'sumă'.

Programul rulează astfel:

- se citesc variabilele a și b;
- se predă controlul procedurii care atribuie variabilei c suma dintre a și b;
- se revine în programul apelant, unde se tipărește valoarea variabilei c.

Din studiul acestui exemplu mai observăm că:

- variabilele a, b,c sunt variabile ale blocului apelant;
- ele sunt recunoscute în procedură (dacă am fi declarat o variabilă în procedură, ea nu ar fi fost recunoscută în programul apelant).

Exemplul 2

```
program pr2;
var a,b,c,d:integer;

  procedure adun;
  begin
    d:=a+b
  end;

  procedure scad;
  begin
    d:=d-c
  end;

begin
  write('a=');
  readln(a);
  write('b=');
  readln(b);
  write('c=');
  readln(c);
```

```
  adun;
  scad;
  writeln('d=',d)
end.
```

În acest exemplu programul calculează expresia a+b−c, prin apelul a două proceduri 'adun' și 'scad'. Observăm că:

- procedurile au fost declarate una după alta;
- variabilele globale ale programului sunt recunoscute de ambele proceduri.

Exemplul 3

```
program pr3;
var a:integer;

  procedure ad;
  var b:integer;
  begin
    write('b=');
    readln(b);
    a:=a+b
  end;

begin
  write('a=');
  readln(a);
  ad;
  write('a=',a)
end.
```

Acest program calculează suma a două numere a și b. Rularea sa decurge în felul următor:

- în programul principal, se citește variabila a;
- se predă controlul procedurii 'ad' care efectuează următoarele:
  - citește variabila b;
  - face suma dintre a și b și atribuie rezultatul variabilei a;
- se revine în blocul apelant unde se tipărește a.

În acest exemplu observăm următoarele:

- în cadrul procedurii am declarat variabila b (locală procedurii dar necunoscută blocului apelant);
- în cadrul procedurii sunt recunoscute ambele variabile.

Exemplul 4

```
program pr4;
var a:integer;

  procedure ad;
  var b:integer;

    procedure sd;
    var c:integer;
    begin
      write('c=');
```

```
      readln(c);
      b:=b-c
    end;

  begin
    write('b=');
    readln(b);
    sd;
    a:=a+b
  end;

begin
  write('a=');
  readln(a);
  ad;
  writeln('a=',a)
end.
```

Programul prezentat calculează expresia a + b–c. Pentru aceasta, programul principal apelează o procedură declarată în cadrul lui, iar aceasta la rândul ei apelează o altă procedură pe care o conține.

Rularea decurge în felul următor:

- se citește a;
- se predă controlul procedurii 'ad';
  - aceasta îl citește pe b și predă controlul procedurii 'sd';
    - aceasta îl citește pe c, calculează b–c și atribuie rezultatul variabilei b;
  - se revine în procedura 'ad' care calculează a + b și atribuie această valoare variabilei a;
- se revine în programul principal, care tipărește conținutul variabilei a.

Exemplul 5

```
program pr5;
var a:integer;

  procedure p1;

    procedure p2;
    begin
      a:=a+1
    end;

  begin
    a:=a+1;
    p2
  end;

begin
  write('a=');
  readln(a);
  p1;
  writeln('a=',a)
end.
```

În acest exemplu se citește o variabilă și se adună cu 2.

Rularea decurge astfel:

- se citește variabila a și se apelează procedura 'p1';
- aceasta adună 1 la variabila a și apelează procedura p2 (pe care o conține);
- procedura p2 adună 1 la variabila a;
- se revine în p1;
- se revine în programul principal care tipărește conținutul variabilei a.

Este interesant de observat faptul că variabila a este recunoscută în 'p2' chiar dacă nu a fost definită în blocul în care s-a făcut declarația acestei proceduri.

Exemplul 6

În acest exemplu se demonstrează cum se redefinește o variabilă. Valoarea tipărită de program va fi 0. Variabila a este declarată de două ori: în programul principal și în procedura 'b'. În acest caz, compilatorul rezervă două spații în memorie: unul pentru variabila a declarată în programul principal, altul pentru variabila a declarată în procedură. Procedura lucrează asupra variabilei declarate în cadrul ei. În acest caz, variabila globală a nu este recunoscută în procedură.

```
program pr6;
var a:integer;

  procedure b;
  var a: integer;
  begin
    a:=1
  end;

begin
  a:=0;
  b;
  writeln(a)
end.
```

Contraexemplu:

```
program pr7;

  procedure a;

    procedure b;
    begin
      writeln('merge');
    end;

  begin
  end;

begin
b
end.
```

Acest program dă eroare de sintaxă. S-a încercat din programul principal apelul unei proceduri ('b') care a fost definită în cadrul procedurii 'a'. Nu ne ajută faptul că am apelat o procedură definită în cadrul unui bloc declarat acolo unde am făcut apelul.

În continuare, se presupun două sau mai multe proceduri declarate în cadrul aceluiași bloc. Se pune întrebarea dacă este posibil să apelăm o astfel de procedură dintr-una declarată la același nivel. Răspunsul comportă o nuanțare.

Dacă procedura apelată este declarată înaintea celei care o apelează, acest lucru este posibil. Programul următor demonstrează acest lucru.

```
program pr8;
var a:integer;

  procedure p1;
  begin
    a:=a+1
  end;

  procedure p2;
  begin
    p1
  end;

begin
  write('a=');
  readln(a);
  p2;
  writeln('a=',a)
end.
```

În situația în care procedura care este apelată se află plasată după procedura care o apelează și dacă nu se iau măsuri suplimentare, acest lucru nu este posibil. Programul următor dă eroare de sintaxă.

```
program pr9;
var a:integer;

  procedure p1;
  begin
    p2
  end;

  procedure p2;
  begin
    a:=a+1
  end;

begin
  write('a=');
  readln(a);
  p1;
  writeln('a=',a)
end.
```

Pentru a putea realiza, din cadrul unei proceduri, apelul unei proceduri plasate după procedura apelantă, antetul procedurii care este apelată se declară la început, urmat de clauza **FORWARD** iar descrierea ei poate fi pusă după procedura apelantă. Programul următor demonstrează acest fapt.

```
program pr9;
var a:integer;

procedure p2; forward;

  procedure p1;
  begin
    p2
  end;

  procedure p2;
  begin
    a:=a+1
  end;

begin
  write('a=');
  readln(a);
  p1;
  writeln('a=',a)
end.
```

6.3.2. Parametri formali, parametri efectivi

A fost arătat faptul că un subprogram poate fi utilizat de mai multe programe. Evident, acesta va lucra cu variabile proprii (de unde solicită datele de intrare, depune rezultatele, face diverse calcule). Ar fi dificil ca toți utilizatorii acestor subprograme să cunoască toate amănuntele (de exemplu că datele de intrare în subprogram se depun în două variabile, una numită d de tipul real, alta b de tip boolean etc.). Apare necesitatea existenței în limbaj a unui mecanism de interfață între subprogram si blocul apelant. Acesta este reprezentat de **parametrii** prin care subprogramul sc mbă informații cu blocul apelant. Ei sunt prezenți atât în antetul subprogramului c ât și în instrucțiunea de apel prezentă în blocul apelant.

Parametrii sit ați în antetul subprogramului se numesc **parametri formali** și sunt alcătuiți c nume de variabile, valori sau alte subprograme. Numele de formali provine din faptul că în momentul declarației nu sunt cunoscuți efectiv (ca valoare). Parametrii situați în instrucțiunea de apel sunt numiți **parametri efectivi** (în momentul apelului valorile acestora sunt cunoscute).

Să analizăm următorul exemplu:

```
program pr12;
var s,s1,n:integer;

  procedure suma(var s:integer;n:integer);
  var i:integer;
```

```
  begin
    s:=0;
    for i:=1 to n do s:=s+i
  end;

begin
  write('n=');
  readln(n);
  suma(s,n);
  writeln('suma =',s);
  suma(s,10);
  writeln('suma primelor 10 numere =',s);
  suma(s1,4);
  writeln('suma primelor 4 numere=', s1)
end.
```

Procedura 'suma' calculează suma primelor n numere naturale. În programul principal ea este apelată de trei ori, în diverse moduri.

Analizăm antetul procedurii. Între paranteze sunt trecuți parametrii formali. Aceștia sunt: var s:integer și n:integer, deci avem doi parametri formali. Ei sunt separați prin semnul ';'. De aici tragem o primă concluzie: parametrii formali sunt separați prin semnul ';'. Dacă urmărim oricare din cele trei apeluri ale acestei proceduri observăm că între paranteze sunt trecuți parametrii efectivi. Avem de fiecare dată câte doi parametri efectivi. Observăm existența egalității între numărul parametrilor formali și numărul parametrilor efectivi. De aici tragem concluzia următoare: numărul parametrilor formali coincide cu numărul parametrilor efectivi. De asemenea observăm că parametrii efectivi sunt separați prin virgulă.

Pentru cele trei apeluri ale procedurii analizăm parametrii efectivi. La primul apel avem s și n, la al doilea – s și 10, la al treilea - s1 și 4. Nu este obligatoriu ca numele parametrilor formali să coincidă cu numele parametrilor efectivi (aceste nume coincid numai la primul apel). Totuși există o altă corespondență și anume de tip. Primul parametru, fie că este formal, fie că este efectiv este de tipul INTEGER, iar al doilea fie că se numește n, fie că este un număr este, de asemenea, de tipul INTEGER.

În concluzie, se pot spune următoarele: între parametrii formali și cei efectivi există o corespondență de număr și tip.

Analizăm din nou parametrii formali. Amândoi sunt de tipul INTEGER. Cu toate acestea, unul a fost declarat cu ajutorul cuvântului VAR, altul fără acest cuvânt. Care este motivul pentru care s-a procedat astfel?

Limbajul Pascal admite două forme de transmitere a parametrilor: prin referință și prin valoare.

6.3.2.1. Transmiterea parametrilor prin referință

În acest caz, un astfel de parametru (formal) este precedat de cuvântul **VAR**. În exemplul nostru este cazul primului parametru. Prin aceasta se precizează că atât subprogramul cât și programul apelant lucrează asupra aceleiași variabile (chiar dacă numele ei este altul). Să fim mai expliciți. Considerăm

apelul suma(s,n). În subprogram, nu se face alocarea de spațiu în memoria calculatorului pentru variabila s. El lucrează în același spațiu alocat pentru s în blocul apelant. Aceasta explică faptul că în blocul apelant regăsim rezultatul final (valoarea lui s) pe care o tipărim. Analizăm apelul suma (s1,4). Cu toate că nu există corespondență de nume (parametrul formal este s iar cel efectiv este s1, atât subprogramul cât și blocul apelant lucrează în spațiul de memorie alocat variabilei s1. Folosim transmiterea parametrilor prin referință, atunci când ne interesează să întoarcem blocului apelant un rezultat.

6.3.2.2. Transmiterea parametrilor prin valoare

În acest caz parametrul formal nu este precedat de cuvântul VAR. Să analizăm exemplul nostru. Este cazul celui de-al doilea parametru. La o astfel de transmisie, subprogramul rezervă un spațiu în memorie pentru această valoare (deci pentru n). Acest fapt ne permite să transmitem ca parametru efectiv chiar o valoare (nu neapărat o varibilă din care, de fapt, este folosit numai conținutul ei). Spre a ne lămuri asupra acestui mecanism considerăm un nou exemplu:

```
program pr13;
var n:integer;

  procedure a(n:integer);
  begin
    n:=n+1;
    writeln(n)
  end;

begin
  n:=1;
  a(n);
  writeln(n)
end.
```

Avem un singur parametru transmis prin valoare. Programul decurge astfel:

- variabilei n i se atribuie valoarea 1;
- este apelată procedura a;
- n se incrementează cu 1 și se tipărește (se va tipări 2);
- se revine în blocul apelant și se tipărește valoarea lui n(1).

Acest program demonstrează că pentru n s-a alocat în subprogram un alt spațiu în memorie. În procedură, valoarea finală pentru n este 2, iar în programul principal este 1 (nu am putut întoarce rezultatul).

Observație: Nu este permis ca variabila ce este trimisă ca parametru să fie de un tip anonim. *Exemplu*: v:array[1..9] of integer nu poate fi trimisă ca parametru. Toate variabilele transmise ca parametru trebuie să aibă declarat tipul utilizând TYPE, ca în exemplul următor:

```
program pr14;
type vector=array[1..9] of integer;
var v:vector;
    n,i,s:integer;
```

```
  procedure suma(n:integer;v:vector;var s:integer);
  begin
    s:=0;
    for i:=1 to n do s:=s+v[i]
  end;

begin
  write('n=');
  readln(n);
  for i:=1 to n do
  begin
    write('v[',i,']=');
    readln(v[i])
  end;
  suma(n,v,s);
  writeln('s=',s);
end.
```

Pentru rigurozitate prezentăm procedurile utilizând diagrame de sintaxă:

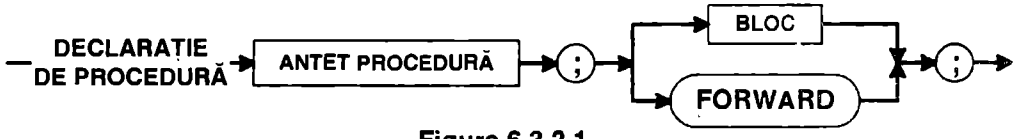


Figura 6.3.2.1.

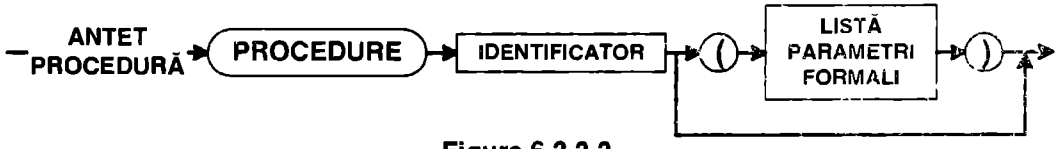


Figura 6.3.2.2.

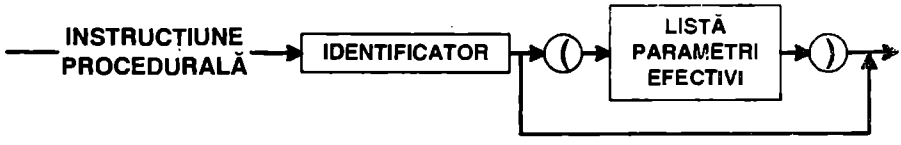


Figura 6.3.2.3.

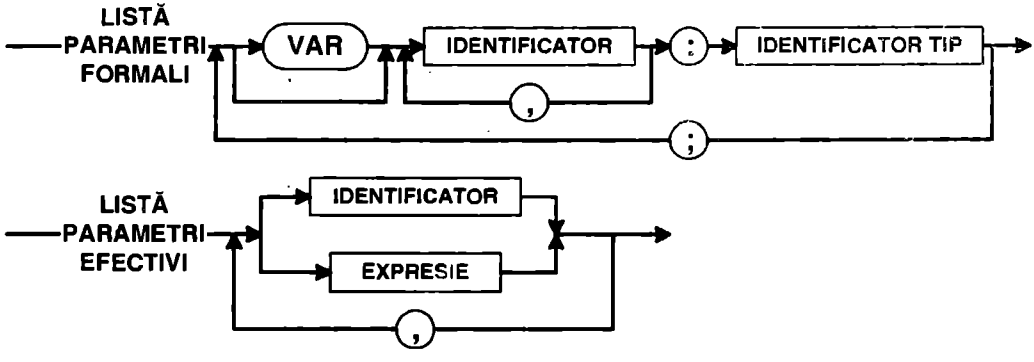


Figura 6.3.2.4.

6.4. Funcții (declarare și apel)

Nu o dată, în matematică intervin expresii de genul: $3*\sin(x)$, $4*\ln(x)+1$ etc. În Pascal este permis ca aceste expresii să fie transcrise direct în limbaj. Pentru a putea face astfel de calcule (sin, ln) se apelează niște subprograme numite **funcții**, care întorc un singur rezultat (astfel de calcule se realizează prin dezvoltări în serie, pe care le vom învăța mult mai târziu). Ne putem pune problema dacă nu este posibil să scriem și noi astfel de subprograme (care să nu depășească cunoștințele noastre de matematică). Răspunsul este afirmativ. Pentru studiul lor pornim de la un exemplu.

```
program pr15;
var s,i:integer;

  function inv(n:integer):integer;
  var ninv:integer;
  begin
    ninv:=0;
    while n<>0 do
    begin
      ninv:=ninv*10+n mod 10;
      n:=n div 10
    end;
    inv:=ninv
  end;

begin
  writeln(inv(125));
  s:=0;
  for i:=15 to 20 do s:=s+inv(i);
  writeln('s=',s)
end.
```

Acest program conține o funcție care are ca parametru de intrare un număr și returnează o valoare întreagă ce reprezintă numărul inversat. Analizând funcția constatăm următoarele: antetul unei funcții cuprinde cuvântul cheie **FUNCTION** urmat de **numele ei**, în continuare se pun **parametrii formali** (pot lipsi), semnul ':' și **tipul funcției** (în cazul nostru INTEGER).

Mecanismul de transmitere a parametrilor este același cu cel de la proceduri deci nu îl vom comenta în acest paragraf.

Tipul funcției dă natura rezultatului. În exemplul nostru, inversul unui număr întreg pozitiv este un întreg pozitiv și de aici rezultă tipul INTEGER.

În afara antetului, numele unei funcții se poate găsi în două locuri: în procedura standard de scriere sau în cadrul unor instrucțiuni de atribuire. Atunci când numele figurează în cadrul unei instrucțiuni de scriere el figurează cu parametru (de exemplu, writeln(inv(125)). În acest caz se apelează funcția cu parametri dați, iar rezultatul este tipărit. Atunci când numele unei funcții se găsește în cadrul unei instrucțiuni de atribuire se disting două cazuri: numele se găsește în partea stângă a instrucțiunii sau în parteą dreaptă a acesteia.

În cazul când numele se găsește în partea dreaptă (în exemplul nostru

(inv:=ninv) funcția se comportă ca o variabilă, în sensul că rezuitatul ei capătă valoarea expresiei din partea stângă (după ce am calculat inversul unui număr în cadrul variabilei inv, funcția ia această valoare).

În situația în care numele se găsește în partea stângă a unei instrucțiuni de atribuire, el trebuie urmat de parametri și atunci funcția se apelează iar rezultatul ei intră în calculul expresiei (în exemplul dat s:=s+inv(i)). În astfel de cazuri se greșește de multe ori în scrierea unei funcții.

Contraexemplu: Rescriem funcția după cum se vede mai jos:

```
function inv(n:integer):integer;
begin
  inv:=0;
  while n<>0 do
  begin
    inv:=inv*10+n mod 10;
    n:=n div 10
  end;
end;
```

Pare mai simplu, nu-i așa? Greșeala în această secvență se găsește la linia: inv:=inv*10+ n mod 10. Numele funcției figurează și în dreapta instrucțiunii de atribuire. Deci se încearcă apelul ei. Dar pentru apel funcției îi sunt necesari parametri pe care nu-i are precizați. În concluzie, compilatorul va genera o eroare de sintaxă.

6.4.1. Tipul funcției

Poate fi de trei feluri:

- orice tip simplu (exemplul anterior);
- de tip reper (vom studia cu altă ocazie acest tip);
- de tip string (exemplul care urmează).

```
program pr16;

  function concat(a,b:string):string;
  begin
    concat:=a+b
  end;

begin
  writeln (concat('studiul ', 'functiilor'))
end.
```

În continuare prezentăm sintaxa funcțiilor utilizând diagramele de sintaxă.

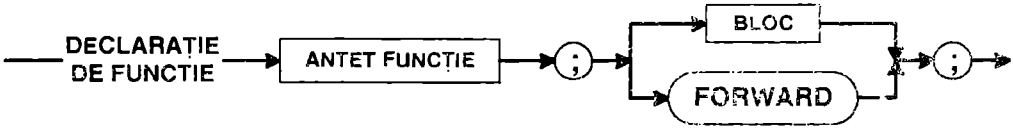


Figura 6.4.1.1.

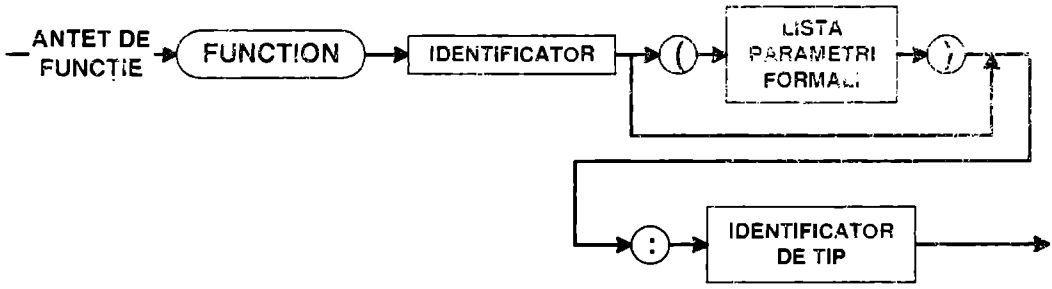


Figura 6.4.1.2.

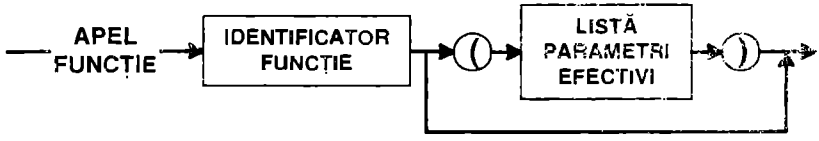


Figura 6.4.1.3.

6.5. Dezvoltare ascendentă, dezvoltare descendentă

Din cele arătate privind apelul procedurilor și funcțiilor, rezultă două modalități de dezvoltare a programelor: dezvoltare ascendentă, dezvoltare descendentă.

6.5.1. Dezvoltarea ascendentă

Programul principal apelează n proceduri. Acestea sunt plasate, una după alta, în partea de declarații (proceduri, funcții) a programului principal. Structura generală este următoarea:

```
program.....
............
procedure ..(     );
procedure ..(     );
.................;
procedure ..(     );
begin
.........
end.
```

Procedurile se pot apela și una pe alta, dacă este îndeplinită una din condițiile:

- procedura care apelează se găsește plasată după procedura apelată;
- nu este îndeplinită condiția anterioară, dar s-a utilizat clauza FORWARD.

Denumirea provine de la faptul că programul principal rezultă, în esență, din apelul acestor proceduri.

6.5.2. Dezvoltarea descendentă

Programul principal apelează o procedură sau funcție, aceasta la rândul ei are declarată o altă procedură sau funcție pe care o apelează, ș.a.m.d. Denumirea provine din faptul că se gândește întregul, după care se vede de ce anume este nevoie pentru realizarea lui (ce procedură sau funcție) apoi acest proces se reia.

Schema generală este următoarea:

```
program.....

  procedure ...(      );
    ..........
      procedure...(    );
      .........
      begin
      .....
      end;
  begin
  .......
  end;

begin
......
end.
```

Aceste modalități de dezvoltare a programelor au un caracter mai mult teoretic, deoarece în practică se folosește o metodă mixtă.

6.6. Unități de program

În activitatea practică se fac aplicații cu un mare număr de programe care de multe ori sunt de mari dimensiuni. O preocupare constantă a informaticienilor este de a pune la dispoziția programatorilor instrumente eficiente de programare. În acest sens, limbajul TURBO PASCAL oferă posibilitatea lucrului cu **unități de program** (unit-uri).

6.6.1. Forma generală și construcția unei unități de program

Prin **unitate de program** înțelegem un ansamblu de date, proceduri și funcții, plasate și compilate împreună care pot fi ușor utilizate în scrierea altor programe, fără a se cunoaște amănunte necesare realizării lor.

Forma generală a unei unități de program este următoarea:

UNIT <nume unitate>;
INTERFACE
[USES <numele altor unități de program pe care le utilizează>;]
[tipuri de date și variabile globale în programul care utilizează unitatea]
antete de proceduri și funcții

IMPLEMENTATION
[tipuri de date și variabile locale (valabile numai pentru unitatea de program)]
procedurile și funcțiile utilizate de unitatea de program

[BEGIN
..... o eventuală secvență de
..... program care se execută
..... în momentul declarării
..... unității de program în
..... programul care o utilizează.
END.]

Se observă că o unitate de program are în mod obligatoriu două părți și opțional o a treia.

Prima parte obligatorie este precedată de **clauza INTERFACE**. Aici se găsesc declarațiile tipurilor de date, variabilelor, și sunt enumerate procedurile și funcțiile ce pot fi utilizate din programul principal. Pentru orice tip de variabilă declarat aici, se pot declara variabile de acest tip atât în cadrul unității cât și în cadrul programului care folosește unitatea. Orice variabilă declarată aici poate fi folosită fără nici o altă declarație atât în cadrul unității cât și în programul care folosește unitatea (unde are rolul de variabilă globală). Toate procedurile și funcțiile care aici sunt numai enumerate pot fi folosite de orice program care folosește unitatea.

A doua parte, obligatorie, se găsește plasată sub **clauza IMPLEMENTATION**. În această parte se pot declara tipurile și variabilele valabile numai pentru unitatea dată. Tot aici se scriu în clar toate procedurile și funcțiile care în prima parte au fost numai declarate, precum și altele ce pot fi folosite numai în cadrul unității.

Partea opțională a fost prezentată în cadrul formei generale a unității.

În continuare, vom da un exemplu pur didactic de unitate de program, numită OPERAȚII. Aceasta conține patru proceduri de adunare, scădere, înmulțire și împărțire de variabile reale. Variabilele a și b sunt globale pentru programul care utilizează această unitate, iar variabila c poate fi utilizată numai în cadrul unității. În secvența de inițializare se citesc valorile variabilelor a și b.

```
unit operatii;
interface
var a,b:real;

procedure adun(a,b:real);
procedure scad(a,b:real);
procedure produs(a,b:real);
procedure impart(a,b:real);

implementation
var c:real;

procedure adun(a,b:real);
begin
  c:=a+b;
  writeln (c:3:2)
end;
```

```
procedure scad(a,b:real);
begin
  c:=a-b;
  writeln(c:3:2)
end;

procedure produs(a,b:real);
begin
  c:=a*b;
  writeln (c:3:2)
end;

procedure impart(a,b:real);
begin
  if b=0  then writeln ('Împartirea nu se poate face ')
          else  begin
                  c:=a/b;
                  writeln(c:3:2)
                end
end;

begin
  write('a='); readln(a);
  write('b='); readln(b);
end.
```

O unitate de program trebuie să se găsească compilată pe disc având **extensia TPU**. Pentru aceasta se procedează în felul următor:

- se tastează F10 și se deplasează cursorul către opțiunile de compilare (sau direct ALT+C)
- se alege opțiunea DESTINATION MEMORY;
- îndată ce suntem poziționați pe această opțiune tastăm D, precizând astfel ca rezultatul compilării să se găsească pe disc;
- pentru declanșarea compilării se tastează F9.

Programul următor utilizează această unitate. Faptul că acest program utilizează o unitate se precizează utilizând **clauza USES**. Rularea sa decurge în felul următor:

- se cer valorile variabilelor a și b (declarate în unitate);
- se apelează procedurile de adunare, scădere, înmulțire și împărțire iar rezultatele se tipăresc;
- se apelează aceleași proceduri utilizând pentru apel două variabile globale ale programului (c și d).

```
program test;
uses operatii;
var c,d:real;

begin
  adun(a,b);
  scad(a,b);
  produs(a,b);
  impart(a,b);
  write('c='); readln(c);
  write('d='); readln(d);
  adun(c,d);
```

```
    scad(c,d);
    produs(c,d);
    impart(c,d);
end.
```

După ce am văzut modul de alcătuire și construcție al unei unități de program, trebuie să precizăm faptul că limbajul Turbo Pascal este livrat cu câteva unități de program standard (produse de firma BORLAND). Amintim câteva: **SYSTEM, CRT, DOS, GRAPH**. Procedurile și funcțiile puse la dispoziție de limbaj le găsim în aceste unități.

6.7. Proceduri și funcții predefinite

În unitatea de program SYSTEM se găsesc majoritatea procedurilor și funcțiilor prezentate în capitolele anterioare.

Exemple de funcții: sin, cos, ln, ord, succ etc.

Exemple de proceduri: delete, insert, read, readln etc.

În acest paragraf prezentăm o procedură și o funcție pentru generarea numerelor aleatoare (întâmplătoare). Acestea sunt extrem de utile atunci când scriem programe pentru diverse jocuri. Mișcarea unei figuri într-o parte sau în alta este consecința generării unui număr aleator.

Procedura **RANDOMIZE** inițializează variabila de pornire a generatorului de numere aleatoare. Această variabilă se numește **RANDSEED** și este declarată în unit-ul SYSTEM.

Funcția **RANDOM(n)** generează un număr aleator cuprins între 0 și n−1. Fără parametru, funcția generează un număr aleator cuprins în intervalul [0,1).

Programul următor realizează un joc. Calculatorul generează un număr aleator format din 4 cifre distincte, cuprinse între 0 și 9. Prima cifră trebuie să fie diferită de 0. Acest număr nu este tipărit. Operatorul va introduce și el un număr care respectă aceleași condiții. Cifrele celor două numere sunt comparate de calculator, care scrie câte din ele coincid ca poziție și valoare și câte coincid numai ca valoare (centrate și necentrate).

Exemplu: calculatorul „a ascuns" numărul 3172. Operatorul tastează 4129. Răspunsul este una centrată (este vorba despre cifra 1, și una necentrată (cifra 2). Procedeul se repetă până când numărul a fost ghicit (4 centrate). Rulați acest program și încercați să ghiciți numărul ascuns din cât mai puține încercări.

```
program centrate;
type numar=array[1..4] of integer;
var nr_ascuns,nr_citit:numar;
    c,n,i,nr:integer;

function nr_corect(nr:numar):boolean;
var i,j:integer;
begin
  nr_corect:=true;
```

```
  for i:=1 to 3 do
    for j:=i+1 to 4 do
      if nr[i]=nr[j] then nr_corect:=false;
  if nr[1]=0 then nr_corect:=false;
end;

procedure c_n(nr1,nr2:numar;var c,n:integer);
var i,j:integer;
begin
  c:=0;
  n:=0;
  for i:=1 to 4 do
    if nr1[i]=nr2[i] then c:=c+1;
  for j:=1 to 4 do
    if (nr1[i]=nr2[j])and(i<>j) then n:=n+1
end;

procedure ascund(var nr:numar);
var i:integer;
begin
  for i:=1 to 4 do nr[i]:=random(10)
end;

begin
  randomize;
  repeat
    ascund(nr_ascuns);
  until nr_corect(nr_ascuns);
  repeat
    repeat
      readln(nr);
      i:=4;
      while nr<>0 do
      begin
        nr_citit[i]:=nr mod 10;
        nr:=nr div 10;
        i:=i-1
      end
    until nr_corect(nr_citit);
    c_n(nr_ascuns,nr_citit,c,n);
    writeln('centrate=',c,' necentrate=',n)
  until c=4
end.
```

6.8. Parametri formali de tip procedură sau funcție

6.8.1. Tipul de date procedural

Printr-un **tip procedural** se descrie totalitatea procedurilor sau funcțiilor cu aceiași parametri formali.

O variabilă de un anume tip procedural poate primi ca valoare o procedură sau funcție. Pentru ca programul să poată lucra cu tipul procedural avem două posibilități:

- Să fie prezentă opțiunea de compilare FORCE FAR CALLS din submeniul COMPILER al meniului OPTIONS.
  - se tastează ALT + O;
  - se deplasează bara de meniu cu ajutorul săgeților pe opțiunea COMPILER:
  - se tastează RETURN (s-a făcut selecția);
  - la opțiunea FORCE FOR CALLS se tastează spațiu (se selectează această opțiune).
- Procedurile sau funcțiile care fac obiectul atribuirii vor fi cuprinse între {$f+} și {$f-}.

În programul care urmează se exemplifică modul de lucru cu tipul procedural.

```
program pro1;
type a=procedure(x:integer);
var p:a;

procedure adun (i:integer);
begin
  i:=i+1;
  writeln(i)
end;

begin
  p:=adun;
  p(3)
end.
```

Tipul a este un tip procedural. El reprezintă totalitatea procedurilor cu un singur parametru formal de tip INTEGER.

Variabila p este de tipul procedural. Valorile ei sunt procedurile specificate de tipul a. În program, se efectuează o instrucțiune de atribuire asupra variabilei p. În urma acestei atribuiri ea capătă ca valoare procedura adun. Procedura este lansată în execuție cu parametrul efectiv 3.

Programul următor exemplifică modul de lucru cu tipul procedural care admite ca valori funcții cu un singur parametru real de tipul INTEGER.

```
program pro2;
type a=function(x:integer):integer;
var p:a;

function adun (i:integer):integer;
begin
  adun:=i+1;
end;

begin
  p:=adun;
  writeln(p(3))
end.
```

Procedurile și funcțiile admit ca parametri formali variabile de un tip procedural.

În exemplul următor, procedura tipar are ca parametru formal de intrare variabila p care poate lua ca valori funcții de tip INTEGER cu un parametru formal de tip INTEGER. Procedura este apelată pentru două funcții diferite (adun și scad) și, în mod evident, rezultatele celor două execuții sunt diferite.

```
program pro3;
type a=function(x:integer):integer;
var p:a;

function adun (i:integer):integer;
begin
  adun:=i+1;
end;
function scad (i:integer):integer;
begin
  scad:=i-1;
end;
procedure tipar(p:a);
begin
  writeln(p(9))
end;

begin
  p:=adun;
  tipar(p);
  p:=scad;
  tipar(p)
end.
```

Se poate pune firească întrebare, cum este posibil ca o variabilă, fie ea și de un tip procedural, să ia valoarea unei proceduri sau funcții? În realitate, o astfel de variabilă ia ca valoare **o adresă** către procedura sau funcția specificată. Atunci când se face apelul procedurii (funcției), prin intermediul adresei se face transferul către aceasta.

În cadrul unităților de program s-a realizat pentru prima dată unitatea între proceduri și date în cadrul unei unice structuri sintactice. Se pune întrebarea dacă această legătură este posibilă și în cadrul unei alte structuri sintactice și anume tipul RECORD. Răspunsul la această întrebare este afirmativ. Tipul RECORD adună în cadrul unei înregistrări date de diverse tipuri printre care și cele de tip procedural (deci și proceduri sau funcții). În programul care urmează, o înregistrare de tip OBIECT conține un număr întreg și o funcție întreagă de parametru formal întreg. Se fac atribuiri pentru două variabile de tip OBIECT și se lansează procedurile conținute de fiecare în parte.

```
program pro1;
type a=function(x:integer):integer;
     obiect=record
               nr:integer;
               functie:a
            end;
var inr,inr1:obiect;
```

```
function adun (i:integer):integer;
begin
  adun:=i+1;
end;

function scad (i:integer):integer;
begin
  scad:=i-1;
end;

begin
  inr.functie:=adun;
  inr.nr:=5;
  inr1.functie:=scad;
  inr1.nr:=3;
  writeln(inr.functie(inr.nr));
  writeln(inr1.functie(inr1.nr))
end.
```

Acest exemplu ne conduce la următoarele concluzii:

- datele și procedurile pot fi reunite în cadrul unei structuri de date de tipul înregistrare (această unificare se numește **ÎNCAPSULARE**);
- întrucât avem posibilitatea să construim o altă structură de tip înregistrare cu una deja construită, noua structură preia toate datele și procedurile vechii structuri (fenomen ce se numește **MOȘTENIRE**). Aceste două caracteristici stau la baza programării pe obiecte.

6.9. Programarea pe obiecte

6.9.1. Logica și mecanismul programării pe obiecte

În scopul declarat ca activitatea de programare să fie accesibilă oricui dorește să o practice, s-a căutat să se pună în corespondență obiectele realității înconjurătoare cu mecanismele existente în cadrul limbajelor de programare, îndeplinindu-se astfel și un deziderat mai vechi al programatorilor.

Exemplu ce justifică necesitatea programării pe obiecte.

Să considerăm un obiect din viața reală și anume un telefon care se caracterizează prin:

1) tip (public sau nu);
2) modul de transmitere a numărului (disc sau tastatură);
3) proveniența (țara în care a fost fabricat);
4) preț;
5) algoritmul de utilizare.

Un program care gestionează un stoc de astfel de telefoane utilizând tipul RECORD, poate reține fără probleme, pentru un telefon, primele patru informații. Pentru a reține algoritmul de utilizare a telefonului avem următoarele

posibilități:

- în funcție de tipul telefonului și modul de transmitere a numărului, în cadrul înregistrării să se memoreze un text care să descrie acest algoritm, soluție extrem de ineficientă;
- programul să conțină o procedură care analizând înregistrarea pentru un telefon să furnizeze algoritmul său de funcționare (în acest caz legătura între procedură și date nu se realizează în cadrul unei construcții sintactice unice, iar în cele ce urmează se vor evidenția și alte dezavantaje).

Aceste neajunsuri încearcă să le rezolve programarea pe obiecte.

Pentru aceasta, limbajul TURBO PASCAL 6.0 dispune de un mecanism specific. Astfel, apare un nou tip și anume **tipul obiect.**

Prin obiect se înțelege o construcție sintactică unică în care se pun la un loc datele și procedurile care lucrează cu aceste date și care este recunoscut de compilator prin cuvântul cheie **OBJECT**.

Procedurile sau funcțiile existente într-un obiect se numesc „**METODE**".

Rezultă că, pentru exemplul nostru, avem nevoie de o variabilă obiect pe care o putem numi TELEFON și care să conțină toate informațiile **1)...4)** precum și metoda (procedura care descrie algoritmul de folosire a acestuia).

În ultimii ani au apărut telefoane moderne cu facilități suplimentare (memorie, robot etc.). Este evident faptul că algoritmul de utilizare a unui astfel de telefon conține în plus și alte elemente cum ar fi memorarea unui număr de telefon, efectuarea unui apel după un număr deja memorat, lansarea unui mesaj pe robot etc. Aceasta înseamnă că algoritmul de folosire a telefonului se complică deci avem nevoie de noi metode pe lângă cele vechi. Rezultă că este absurd să rescriem aceste proceduri într-un alt tip de obiect TELEFON_M. În cadrul limbajului există posibilitatea de a construi un nou obiect (în cazul nostru TELEFON_M) pornind de la unul vechi, preluând toate datele și metodele acestuia la care se pot adăuga date și/sau metode noi. Revenind la ipoteza în care nu folosim programarea pe obiecte și avem o procedură din exterior care furnizează algoritmul, aceasta ar trebui modificată, fapt ce presupune un efort suplimentar de programare.

6.9.2. Mecanismul de realizare a programării pe obiecte

6.9.2.1. Sintaxa tipului obiect și încapsularea

Sintactic, structura tip obiect se poate prezenta în două feluri:

1.
```
OBJECT
metoda_1;
{metoda_2};
.....
{metoda_n}
PRIVATE
```

```
lista_1 câmpuri :tip;
{lista_2 câmpuri :tip;
.....
{lista_p câmpuri :tip}
end
```

2.
```
object
lista_1 câmpuri :tip;
{lista 2 câmpuri :tip;}
.....
{lista_p câmpuri :tip:}
metoda_1;
{metoda_2};
.....
{metoda_n}
end
```

În cazul formei **1.** cuvântul **PRIVATE** specifică faptul că ceea ce urmează între el și **END** nu este accesibil decât metodelor asociate tipului obiect care îl conține. Precizăm de asemenea că din punct de vedere al limbajului, putem să nu folosim acest cuvânt și deci să avem acces la datele unei variabile obiect și din cadrul programului, însă acest lucru nu este permis din punctul de vedere al programării pe obiecte (accesul la date se face numai prin intermediul metodelor). Tot astfel limbajul permite utilizarea lui GO TO însă acest lucru este nepermis din punctul de vedere al programării structurate.

Forma **2.** este utilă în situația în care se folosește un obiect spre a declara un alt obiect care îl moștenește pe acesta și posedă metode noi, care au acces în câmpurile de date ale vechiului obiect. În acest caz, sarcina programatorilor este să nu acceseze datele obiectului din programul care îl folosește pe acesta.

Să considerăm un obiect care apare în multe programe și anume un vector de numere întregi. Acesta se caracterizează prin următoarele:
- numărul de componente (să numim acest număr lungimea vectorului);
- vectorul propriu-zis;
- o procedură care permite citirea lungimii și a vectorului de la tastatură (cit);
- o procedură ce permite tipărirea pe monitor a vectorului (tip).

Toate acestea se reține în variabila obiect VECTOR. Variabila obiect împreună cu procedurile aferente se reține într-o unitate de program care, în exemplul nostru, se numește OBIECT1.

Precizăm următoarele reguli:
- variabila obiect se reține în partea de interfață a unității de program (în cazul nostru se numește VECTOR);
- descrierea în clar a procedurilor existente (metodelor) într-o variabilă obiect se realizează în partea de implementare a unității de program;
- în partea de implementare, numele fiecărei proceduri menționat în cadrul variabilei obiect se formează din numele variabilei obiect urmat de punct și numele procedurii existent în cadrul variabilei obiect (exemplu: procedura numită cit în cadrul variabilei obiect se numește aici vector.cit).

În continuare prezentăm unitatea de program care are descrisă variabila obiect VECTOR.

```
unit obiect1;

interface
type vector=object
              procedure cit;
              procedure tip;
              private
              lung:integer;
              vec:array[1..20] of integer;
            end;

implementation

procedure vector.cit;
var i:integer;
begin
  write('lung=');
  readln(lung);
  for i:=1 to lung do
  begin
    write('elem[',i,']=');
    readln(vec[i])
  end
end;

procedure vector.tip;
var i:integer;
begin
  for i:=1 to lung do
  begin
    write('elem[',i,']=',vec[i]);
    writeln
  end
end;
end.
```

Primul exemplu prezentat arată modul în care un program utilizează un obiect. Programul citește un vector și îl tipărește. Regulile care stau la baza alcătuirii unui astfel de program sunt următoarele:

- se precizează numele unității de program ce conține obiectul utilizând clauza USES;
- se declară variabilele care sunt utilizate în program (în cazul nostru utilizăm o singură variabilă și anume a);
- apelul unei metode asociate variabilei se face prin numele variabilei urmat de punct și numele metodei (exemplu: pentru variabila a procedura cit se apelează prin a.cit).

```
program exemplu1;
uses obiect1;
var a:vector;
begin
```

```
  a.cit;
  a.tip;
end.
```

Până aici, am arătat modul în care se realizează „ÎNCAPSULAREA".

Prin „ÎNCAPSULARE" înțelegem înglobarea în cadrul unei structuri sintactice unice atât a datelor cât și a procedurilor (metodelor) ce prelucrează aceste date. Ideea de încapsulare nu apare prima dată în cazul programării pe obiecte, ea a fost întâlnită în cadrul unităților de program și în cadrul tipurilor RECORD ce conțin pe lângă date și tipuri procedurale.

Variabila a de tip vector cuprinde lungimea vectorului, vectorul propriu-zis precum și două proceduri de scriere și citire a acestui vector.

6.9.2.2. Moștenirea

În cele ce urmează exemplificăm mecanisul de **„MOȘTENIRE"**. Să presupunem că ne interesează o situație privind notele obținute de elevii unui liceu la un examen (notele sunt considerate numere întregi). În acest exemplu avem o nouă variabilă obiect (ELEV) care trebuie să conțină :

- numele elevului;
- o procedură de citire a acestui nume;
- notele elevului și procedura de citire a acestora.

Notele obținute de elev la examen constituie un vector. Pentru acesta avem deja un obiect (VECTOR) ce include și metoda de citire a componentelor vectorului (a notelor obținute de elev). Rezultă că obiectul VECTOR ne este de mare folos în construirea noului obiect ELEV. Limbajul permite acest lucru. Analizând unitatea de program în care construim acest nou obiect se desprind următoarele :

- noua unitate de program OBIECT2 utilizează unitatea OBIECT1, acest lucru se specifică utilizând clauza USES.
- noul obiect îl utilizează pe cel vechi și acest lucru se specifică plasând numele obiectului utilizat, așezat între paranteze, după cuvântul cheie OBJECT (în acest mod, noul obiect preia toate datele și metodele vechiului obiect);
- datele și metodele care apar pentru prima dată aici se descriu în clar în noul tip după regulile deja specificate.

```
unit obiect2;

interface
uses obiect1;
type elev=object(vector)
             procedure cit_nume;
             private
             nume:string[20];
           end;

implementation
```

```
procedure elev.cit_nume;
begin
  write('nume elev=');
  readln(nume)
end;
end.
```

În exemplul care urmează se utilizează o variabilă obiect de tip ELEV. Pentru un elev se citesc numele, numărul de note și notele. Se observă că procedura cit a fost moștenită de la tipul vector.

```
program exemplu2;
uses obiect2;
var a:elev;
begin
  a.cit_nume;
  a.cit
end.
```

Prin „MOȘTENIRE" se înțelege acea proprietate a obiectelor prin care un obiect nou construit preia datele și metodele unui alt obiect.

Observație foarte importantă:
Să urmărim o consecință a fenomenului de „MOȘTENIRE". Am construit tipul vector. Cu ajutorul lui am construit tipul elev. În programare vectorii sunt mult utilizați. Cu tipul vector se poate construi un alt obiect cum este acela care reține o mulțime de numere întregi (m_int care, spre deosebire de vector, are și o metodă de verificare a elementelor distincte). Cu acest din urmă obiect se pot construi alte obiecte. Toate acestea preiau rutinele pentru vector. În acest mod obținem reutilizarea softului deja realizat iar un program (care se scrie relativ ușor dar preia soft prin „MOȘTENIRE") poate avea efecte spectaculoase.

6.9.2.3. Atribuirea în cazul variabilelor obiect

Instrucțiunea de atribuire funcționează și cu variabile obiect. De exemplu dacă A și B sunt două variabile de tip VECTOR se poate folosi instrucțiunea A:=B.

Observație foarte importantă:
Putem atribui unei variabile obiect, de un tip ce a fost folosit în construcția altui tip, valoarea unei variabile din noul tip. De exemplu, unei variabile de tip vector i se poate atribui valoarea unei variabile de tip elev.
Programul care urmează exemplifică acest lucru. Se citesc numele și notele unui elev în variabila b de tip elev. Variabilei a de tip vector i se atribuie valoarea variabilei b de tip elev (deci a va avea ca valori notele elevului). Faptul că operația de atribuire a reușit se probează în program prin tipărirea vectorului.

```
program exemplu3;
uses obiect2,obiect1;
var a:vector;
```

```
    b:elev;
begin
  b.cit;
  b.cit_nume;
  a:=b;
  a.tip
end.
```

Observație: Atribuirea unei variabile obiect de un tip descendent a valorii unei variabile de un tip strămoș nu este posibilă. De exemplu, unei variabile de tip elev nu i se poate atribui valoarea unei variabile de tip vector. Care este rațiunea acestui fapt?

În ipoteza că această atribuire este posibilă, rămân câmpuri necompletate în variabila de un tip descendent. Ce semnificație logică poate avea conținutul unei variabile de tip elev care conține un elev fără nume dar cu note?

6.9.2.4. Proceduri cu parametri formali de tip obiect

Variabilele de tip obiect pot fi parametri pentru proceduri la fel ca și celelalte variabile. De asemenea, dacă în fraza PROCEDURE apare un parametru de un tip strămoș al acestuia, acestei proceduri i se poate transmite o variabilă de un tip descendent din acesta. În exemplul care urmează procedurilor test1, test2 li se pot trimite ca parametri variabile de tip vector și variabile de tip elev. Ca și în cazul atribuirii, invers nu este posibil (unui parametru de un tip descendent nu i se poate atribui o variabilă de un tip strămoș).

Această proprietate are consecințe practice deosebite. Să presupunem că am construit o procedură ce lucrează cu niște variabile obiect. Construim alte obiecte descendente din obiectul care dă tipul variabilelor cu care lucrează această procedură. Procedura va putea lucra și cu variabile din tipul obiectelor descendente. Iată cum o procedură poate lucra și cu variabile de alt tip decât acelea pentru care a fost concepută.

```
program exemplu4;
uses obiect1,obiect2;
var a:vector;
    b:elev;

procedure test1(var m:vector);
begin
  m.cit
end;

procedure test2(m:vector);
begin
  m.tip
end;

begin
  test1(a);
  test2(a);
  test1(b);
```

```
  test2(b)
end.
```

6.9.2.5. Polimorfism

În biologie, prin polimorfism se înțelege fenomenul prin care indivizi aparținând aceleiași specii apar sub forme diferite.

Prin analogie, în programarea pe obiecte, prin **polimorfism** se înțelege că o procedură, recunoscută printr-un nume, poate să apară sub mai multe forme (mai multe proceduri cu același nume).

În procesul de „MOȘTENIRE" poate apărea problema redefinirii unor metode (pentru un obiect descendent dorim înlocuirea unei metode a unui obiect strămoș cu o metodă nouă). Pentru aceasta, în noul obiect se scrie o metodă cu același nume ca acela al metodei ce se înlocuiește.

În exemplul următor vom construi două obiecte număr și număr1. Pentru obiectul număr1 am redefinit procedura tip (modificând mesajul la afișare).

```
unit obpol1;

interface
type numar=object
            nr:integer;
            procedure cit;
            procedure tip;
          end;

implementation

procedure numar.cit;
begin
  write('numar=');readln(nr)
end;
procedure numar.tip;
begin
  writeln('numar=',nr)
end;
end.

unit obpol2;

interface
uses obpol1;
type numar1=object(numar)
              procedure tip;
            end;

implementation

procedure numar1.tip;
begin
  writeln('numar1=',nr);
end;
end.
```

Programul care urmează lucrează cu cele două obiecte. Atunci când se apelează metoda tip se selectează automat cea specifică obiectului. Când se apelează metoda pentru un obiect, numele ei este căutat în cadrul obiectului. Dacă este găsită se apelează (cazul nostru), în caz contrar se caută metoda într-un obiect ascendent până este găsită.

Pentru programul nostru, dacă se citește 20 se tipărește "număr=20" și apoi, dacă se citește 23, se tipărește "număr1=23".

```
program tpol;
uses obpol1,obpol2;
var a:numar;
    b:numar1;
begin
  a.cit;
  a.tip;
  b.cit;
  b.tip
end.
```

În continuare, se construiesc din nou două obiecte numite tot număr și număr1. Deosebirea constă în faptul că obiectul număr conține o nouă metodă adun. Aceasta adună 1 la câmpul nr al obiectului, după care se apelează procedura tip.

```
unit obpol11;

interface
type numar=object
            nr:integer;
            procedure cit;
            procedure tip;
            procedure adun;
          end;

implementation

procedure numar.cit;
begin
  write('numar=');readln(nr)
end;

procedure numar.tip;
begin
  writeln('numar=',nr)
end;

procedure numar.adun;
begin
  nr:=nr+1;
  tip
end;
end.

unit obpol21;
```

```
interface
uses obpol11;
type numar1=object(numar)
               procedure tip;
             end;

implementation

procedure numar1.tip;
begin
  writeln('numar1=',nr);
end;
end.
```

În programul care urmează se declară o variabilă de tip număr1. După citirea câmpului nr al obiectului se observă că tipărirea se face după metoda tip a obiectului număr. Este firesc să fie așa deoarece metoda adun care apelează tip se găsește în cadrul obiectului număr deci se apelează metoda tip din cadrul obiectului de unde se face apelul.

De exemplu, dacă se citește 3, se tipărește "număr=4".

```
program tpol;
uses obpol21;
var b:numar1;
begin
  b.cit;
  b.adun
end.
```

Se pot pune două întrebări:

1. Care este motivul pentru care în cazul metodelor redefinite se apelează acea metodă care se găsește în obiectul din care s-a făcut apelul sau într-un obiect ascendent?
2. Cum este posibil ca în această situație să putem noi selecta care din variantele redefinite să fie apelată ?

Răspunsul pentru prima întrebare constă în afirmația că, aici, s-a folosit o „METODĂ STATICĂ" de apel. La momentul compilării unităților de program nu se știa care metodă se va apela, așa că s-a apelat acea metodă care se găsește în obiectul de unde s-a făcut apelul iar dacă metoda nu se găsește acolo, se caută în primul ascendent ș.a.m.d.

În continuare răspundem la a doua întrebare.

Există și o „METODĂ DINAMICĂ sau VIRTUALĂ" de apel care permite ca selecția unei metode sau a alteia din cele redefinite să se facă în momentul apelului și nu la compilare. Pentru a realiza aceasta trebuie să procedăm astfel:

- când se declară, în cadrul obiectului strămoș, o metodă ce va fi redefinită, se adaugă după declarație cuvântul **VIRTUAL**;
- orice redefinire a metodei, în cadrul obiectelor ascendente, se va face utilizând același cuvânt cheie **VIRTUAL**;
- fiecare obiect ce conține o metodă redefinită va avea o metodă specifică numită **CONSTRUCTOR**.

Metoda CONSTRUCTOR este prima care se va apela din cadrul obiectului și are un dublu rol:

- după apelul ei, relativ la un obiect, o metodă care apare redefinită este apelată din cadrul obiectului pentru care s-a făcut apelul constructorului și nu din cadrul obiectului din care s-a făcut apelul;
- în cadrul metodei CONSTRUCTOR se pot face opțional și alte operații cum ar fi inițializări etc., motiv pentru care în antetul unui constructor se pot pune parametri ca la orice altă metodă.

Observații:

- metodele CONSTRUCTOR nu pot fi redefinite;
- datele din cadrul obiectelor nu pot fi redefinite.

În continuare se prezintă cele două obiecte număr și număr1 realizate după cele spuse mai sus.

```
unit obpol111;

interface
type numar=object
            nr:integer;
            constructor init;
            procedure cit;
            procedure tip;virtual;
            procedure adun;
          end;

implementation

constructor numar.init;
begin
end;

procedure numar.cit;
begin
  write('numar=');
  readln(nr)
end;

procedure numar.tip;
begin
  writeln('numar=',nr)
end;

procedure numar.adun;
begin
  nr:=nr+1;
  tip
end;
end.

unit obpol211;

interface
uses obpol111;
type numar1=object(numar)
              constructor init;
              procedure tip;virtual;
```

```
            end;

implementation

constructor numar1.init;
begin
end;

procedure numar1.tip;
begin
  writeln('numar1=',nr);
end;
end.
```

În programul care urmează se verifică cele prezentate. Metoda adun apelează metoda cit a obiectului număr1.

```
program tpol2;
uses obpol211;
var a:numar1;
begin
  a.init;
  a.cit;
  a.adun
end.
```

Programul următor verifică faptul că după apelul unui alt constructor se anulează efectul primului apel de constructor și se selectează metoda redefinită corespunzătoare obiectului pentru care s-a făcut ultimul apel.

```
program tpol3;
uses obpol111,obpol211;
var a:numar;
    b:numar1;
begin
  a.init;
  a.cit;
  a.adun;
  b.init;
  b.cit;
  b.adun
end.
```

6.10. Aplicații la capitolul 6

6.10.1. Proceduri, funcții, unități de program, obiecte

Pentru toate problemele în care se solicită scrierea unei proceduri sau funcții se va da și un exemplu de apel al acestora din cadrul programului principal.

1. Să se scrie o procedură care adună două numere reale.

2. Să se scrie o funcție de tip întreg care înmulțește două numere întregi.

3. Să se scrie o procedură care adună doi vectori cu același număr de elemente și întoarce rezultatul într-un vector.

4. Să se scrie o funcție care returnează maximul dintr-un vector de numere reale.

5. Să se scrie o procedură care sortează crescător elementele de pe diagonala principală a unei matrice pătratice.

6. Să se scrie o funcție booleană care returnează TRUE dacă un element citit se găsește într-un vector sortat crescător.

7. Să se citească două matrice și să se facă suma lor. Programul se va realiza astfel:
- se scrie o procedură de citire a unei matrice cu m linii și n coloane;
- se scrie o procedură de tipărire a unei matrice cu m linii și n coloane;
- se scrie o procedură care adună două matrice;
- programul principal rezultă din apelul acestor proceduri.

8. Să se calculeze coeficienții polinomului $P(x) = (x + a)^n$. Programul va utiliza o procedură care înmulțește un polinom oarecare de grad k (coeficienții dați într-un vector) cu polinomul $x + a$.

9. Se citesc m și n întregi. Să se tipărească fracția ireductibilă m/n. Programul va utiliza o funcție de tip INTEGER care returnează c.m.m.d.c. a două numere.

10. Să se tipărească toate numerele prime aflate între doi întregi citiți. Programul va folosi o funcție booleană care returnează dacă un număr este prim sau nu.

11. Scrieți un program care tipărește numerele întregi găsite între două valori citite care se divid cu suma cifrelor lor. Programul va utiliza o funcție care returnează suma cifrelor unui număr întreg primit ca parametru.

12. Să se scrie o procedură care permută două linii ale unei matrice pătratice.

13. Să se scrie o procedură care permută două coloane ale unei matrice pătratice.

14. Să se scrie un program care sortează descrescător elementele situate pe diagonala secundară a unei matrice pătratice prin permutări de linii și coloane. Programul va utiliza procedurile de la problemele 12 și 13.

15. Scrieți o procedură care sortează descrescător un vector.

Problemele 16-19 vor fi rezolvate exclusiv prin folosirea de proceduri și funcții. Mai clar, problema se descompune în subprobleme. Fiecare subproblemă va fi rezolvată de o procedură sau funcție. Programul principal va rezulta din apelul acestor proceduri și funcții. *Exemplu* problema 16.

16. Se citesc două polinoame de grade m și n (coeficienții fiecărui polinom se citesc într-un vector). Să se afișeze coeficienții polinomului sumă.

Pentru a realiza programul sunt necesare următoarele proceduri:
- procedura de citire a unui polinom de parametri n (gradul polinomului) și p

(vectorul în care se citesc coeficienții);
- procedura de adunare a două polinoame (de fapt se adună coeficienții) cu parametri formali m și n (gradele celor două polinoame), p, q, r, trei vectori cu semnificația de polinoame (r polinomul rezultat);
- procedura de tipărire a unui polinom.

Programul principal realizează următoarele:
- citește gradul primului polinom și apelează procedura de citire a polinomului;
- citește gradul celui de-al doilea polinom și apelează procedura de citire a polinomului;
- apelează procedura de adunare a două polinoame;
- apelează procedura de tipărire a rezultatului.

Observație. Este bine să ne obișnuim să rezolvăm problemele ca în exemplul prezentat. Problema pe care o avem de rezolvat se descompune în subprobleme care se rezolvă cu ajutorul procedurilor sau funcțiilor. Dacă o subproblemă este dificilă se descompune și ea în alte subprobleme. Această metodă se numește TOP DOWN.

17. Citiți două polinoame de grade m și n și calculați polinomul produs.

18. Citiți un polinom F de gradul n și m valori reale a1, a2, ..., am. Calculați valoarea polinomului pentru cele m valori citite.

19. Se citesc 3 valori a, b, n (a și b reale, n întreg). Se cere să se calculeze valoarea unei funcții F (definită pe intervalul [a,b]) în cele n + 1 puncte rezultate din împărțirea intervalului [a,b] în n părți egale. Funcția F va fi dată sub forma unui subprogram de tip funcție.

20. Să se creeze o unitate de program care conține 3 subprograme de tip funcție. Fiecare din aceste subprograme calculează valoarea următoarelor funcții (definite pe R) într-un punct :

a) $$F(x)=\begin{cases} x+1, \text{ dacă } x \le -3; \\ 3*x, \text{ dacă } -3 < x <= 100; \\ 5, \text{ dacă } x > 100 \end{cases}$$

b) $G(x) = \sin(x) + \cos(x)*\cos(2*x)$;

c) $H(x) = 10*\{x\}$ (am notat prin $\{x\}$ partea fracționară a lui x).

21. Să se scrie o procedură care calculează valorile uneia din cele trei funcții din problema 20 în cele n + 1 puncte rezultate din împărțirea intervalului [a,b] în n părți egale.

Indicație. Procedura va avea un parametru formal de tip funcție.

22. Să se scrie o unitate de program care conține următoarele proceduri și funcții utile în calculul cu matrice:
- procedură de citire a unei matrice cu m linii și n coloane;
- procedură de tipărire a unei matrice cu m linii și n coloane;

- procedură de interschimbare a două linii ale unei matrice;
- procedură de interschimbare a două coloane ale unei matrice;
- funcție întreagă care întoarce numărul componentelor pozitive ale unei matrice;
- funcție reală care întoarce valoarea maximă dintre toate componentele unei matrice.

Unitatea va conține și definiția tipului matrice.

23. Scrieți un program care utilizează o parte din subprogramele din problema anterioară. Apelarea acestora se va face din cadrul unității de program.

24. Scrieți un program care creează o înregistrare (de tip RECORD) care are următorul conținut:
- nr de tip integer (reține un număr întreg);
- o variabilă de tip procedură numită CIT care va conține o procedură ce citește un număr întreg (de fapt adresa către o astfel de procedură);
- o variabilă de tip procedură numită TIP care conține o procedură ce tipărește un număr întreg.

Considerați două variabile de tipul descris anterior. Cu ajutorul lor citiți și tipăriți două numere întregi. Nu găsiți o mare asemănare între acest mod de lucru și variabilele obiect? Și o variabilă de tipul de date creat acum și o variabilă de tip obiect conțin date și proceduri (metode).

25. Creați un obiect numit MATRICE. Acesta va conține următoarele date și metode:
- N - numărul de linii ale matricei; M - numărul de coloane ale matricei;
- MAT - o structură de tip matrioe cu elemente numere reale;
- CIT - o metodă care citește o matrice;
- TIP - o metodă care tipărește o matrice.

Observație. Cele două metode vor fi preluate din unitatea de program creată la problema 22. De fapt, în dezvoltarea aplicațiilor se scriu întâi unități de program care conțin diverse subprograme utile și cu ajutorul acestora se scriu diverse obiecte, care apoi se moștenesc, ș.a.m.d.

26. Scrieți o procedură care adună două variabile de tip MATRICE.

27. Creați un obiect numit ABSOLVENT care conține următoarele:
- Numele absolventului (string[30]);
- Situația școlară - o matrice în care liniile reprezintă cei 4 ani de studiu iar coloanele notele obținute de acesta la diverse materii, în ordinea în care acestea au fost trecute în catalog;
- o metodă în care se citește situația școlară;
- o metodă de tipărire a situației;
- o metodă care calculează media generală pe diverși ani de studiu;
- o metodă care calculează media generală;
- o metodă care validează notele (să fie între 5 și 10).

Se cere ca acest obiect să moștenească obiectul MATRICE.

28. Înzestrați obiectul creat la problema 27 cu o altă metodă de tipărire a situației școlare (mai elegantă decât tipărirea unei matrice) și care are același nume cu metoda care tipărește matricea. Cum putem selecta după dorință, o

metodă sau alta de tipărire?

29. Creați un obiect numit NUMAR_MARE. Acesta va conține următoarele:
- un vector care reține numărul (pe cifre);
- o procedură care citește de la tastatură (pe cifre) un astfel de număr;
- o procedură care tipărește un astfel de număr;

30. Scrieți o procedură care are ca parametri formali trei variabile de tip NUMAR_MARE. Procedura adună două numere mari și returnează rezultatul.

31. Scrieți o procedură care înmulțește un NUMAR_MARE cu un număr între 0 și 9.

32. Utilizând procedura de la problema 30, scrieți o procedură care înmulțește conținutul a două variabile de tip NUMAR_MARE.

33. Cum ați concepe și programa un obiect numit POLINOM, în care coeficienții acestuia să fie numere reale?

Capitolul 7

Unitatea de program CRT

7.1. Memoria video

De regulă, informațiile pe care ni le trimite calculatorul se afișează pe monitor. Ne-am întrebat „oare de unde se preiau aceste imagini?" În cele ce urmează răspundem la această întrebare.

Există o memorie cu un rol special numită **memorie video**. Monitorul extrage în permanență din această memorie imaginile pe care le reprezintă. Orice modificare survenită în conținutul memoriei video se reflectă pe ecran. Memoria video se află plasată fizic pe **adaptor**. Calculatoarele compatibile IBM utilizează mai multe adaptoare (CGA, EGA, VGA). Întrucât și cele mai noi adaptoare suportă **modul de lucru CGA** (Color Graphics Adapter), acesta va fi luat drept model în cadrul acestui capitol.

Interfața CGA lucrează în două regimuri: **text** și **grafic**. În acest capitol ne ocupăm exclusiv de regimul text. În acest regim se pot afișa numai caractere. Adresa memoriei video este $b800:0. Cunoașterea acestei adrese permite programatorului să lucreze direct în memoria video (cu alte cuvinte să își gestioneze singur imaginea). Ecranul văzut de noi se împarte în 25 de linii și 80 de coloane. Orice caracter se afișează într-o anumită linie și o anumită coloană. În memoria video, pentru fiecare caracter se utilizează doi octeți. Într-unul din ei se reține **codul caracterului** iar în celălalt se reține **atributele grafice**. Ce înțelegem prin acestea?

Spațiul de afișare a unui caracter (un pătrățel) are o anumită culoare (se numește **culoare de fond**). La rândul său, și caracterul se afișează cu o anumită culoare. Informațiile referitoare la aceste culori se găsesc reținute pentru fiecare caracter într-un octet. Structura acestuia (pe biți) este următoarea: BFFFCCCC. Cei 4 biți, notați cu C, reține culoarea de afișare a caracterului. Rezultă că avem la dispoziție 16 culori, codificate de la 0 la 15. Culoarea de fond se reprezintă pe 3 biți notați F (avem la dispoziție 8 culori de fond, codificate de la 0 la 7). Bitul B indică afișarea continuă (valoarea 0) sau intermitentă (valoarea 1). Orice culoare rezultă ca amestec de roșu, verde și albastru, iar pentru culorile caracterelor poate diferi și intensitatea.

Culorile ce pot fi utilizate se găsesc în tabelul de mai jos:

Culoarea	Valoare	Intens	Roșu	Verde	Albastru
Negru (Black)	0	0	0	0	0
Albastru (Blue)	1	0	0	0	1

Culoarea	Valoare	Intens	Roșu	Verde	Albastru
Verde (Green)	2	0	0	1	0
Turcoaz (Cyan)	3	0	0	1	1
Roșu (Red)	4	0	1	0	0
Violet (Magenta)	5	0	1	0	1
Maro (Brown)	6	0	1	1	0
Gri deschis (LightGray)	7	0	1	1	1
Gri închis (DarkGray)	8	1	0	0	0
Bleu (LightBlue)	9	1	0	0	1
Vernil (LightGreen)	10	1	0	1	0
Bleu deschis (LightCyan)	11	1	0	1	1
Roz (LightRed)	12	1	1	0	0
Violet deschis (LightMagenta)	13	1	1	0	1
Galben (Yellow)	14	1	1	1	0
Alb (White)	15	1	1	1	1

Fondul poate avea numai primele 8 culori din tabel.

Pentru a putea lucra direct în memoria video, o declarăm ca masiv cu 25 de linii, 80 de coloane, și doi octeți pentru fiecare caracter, cerând în același timp ca spațiul pentru această variabilă să fie în zona de memorie video (clauza absolute).

În programul următor întreaga memorie video se umple cu 'x'. Observați efectul pe ecran.

```
program mv1;
type mv =array[1..25,1..80,1..2] of char;
var ecran: mv  absolute $b800:0;
    i,j:integer;

begin
  for i:=1 to 25 do
    for j:=1 to 80 do ecran[i,j,1]:='x'
end.
```

De multe ori este necesar să adresăm octeții memoriei video în două moduri:

- ca octeți care rețin caractere;
- ca octeți care rețin numere.

Din acest motiv se face o dublă declarare a memoriei video (este numai una din modalități). În programul următor, se umple memoria video cu spații (pe fond negru) și se scrie în prima linie și prima coloană caracterul 'a'. Scrierea se face negru, pe fond alb. Pentru această scriere, octetul de atribute culoare ia valoarea 112 (01110000).

```
program mv2;
type  mv=array[1..25,1..80,1..2] of char;
      mvb=array[1..25,1..80,1..2] of byte;
var   ecran: mv  absolute $b800:0;
      ecran1:mvb absolute $b800:0;
      i,j:integer;

begin
   for i:=1 to 25 do
      for j:=1 to 80 do
      begin
         ecran1[i,j,2]:=0;
         ecran[i,j,1]:=' ';
         end;
   ecran1[1,1,2]:=112;
   ecran[1,1,1]:='a';
end.
```

7.2. Ferestre

O fereastră reprezintă o porțiune dreptunghiulară a ecranului în care se fac citiri și scrieri. Întreg ecranul este o fereastră implicită (dacă nu se declară alte ferestre). Procedurile și funcțiile unității CRT se referă la o anumită fereastră (cea care este activă la un moment dat).

Observație: o singură fereastră este activă la un anumit moment.

Deschiderea unei ferestre se face utilizând procedura **WINDOW** care are forma următoare:

WINDOW (x1,y1,x2,y2).

Perechile (x1,y1), (x2,y2) reprezintă coordonatele colțurilor din stânga sus și dreapta jos. Este de menționat faptul că x1, x2 reprezintă coloane și y1,y2 reprezintă linii. Colțul din stânga sus al unei ferestre are coordonatele (1,1).

În programul f1, deschidem o fereastră.

```
program f1;
uses crt;

begin
   window (10,10,20,20)
end.
```

Dacă vom analiza efectul acestui program, vom rămâne surprinși de faptul că fereastra nu este vizibilă. Cu toate acestea cursorul este poziționat în colțul din stânga sus al ferestrei deschise.

Faptul că fereastra deschisă este invizibilă se explică prin culoarea ei de

fond care este aceeași cu cea a ferestrei inițiale.

Pentru precizarea culorii de fond se folosește procedura **TEXTBACKGROUND**.

TEXTBACKGROUND (culoare)

Aceasta are un singur parametru formal, și anume culoarea. Culoarea poate fi dată în două moduri: printr-o cifră cuprinsă între 0 și 7 (vezi tabelul din paragraful anterior), sau printr-un cuvânt în engleză ce exprimă culoarea (prezent în același tabel). Această ultimă formă de exprimare a culorii este posibilă întrucât, în unitatea CRT, numele culorilor au fost declarate drept constante cu valorile codurilor respective. În programul f2 deschidem o fereastră la care stabilim culoarea de fond.

```
program f2;
uses crt;

begin
   window (10,10,20,20);
   textbackground(0)
end.
```

După rularea lui, nu observăm nimic diferit față de modul în care a decurs execuția în programul anterior. Culoarea de fond devine vizibilă în două cazuri:

- după rularea procedurii CLRSCR;
- după ce am scris efectiv în fereastră.

Procedura **CLRSCR** nu are parametri și are următoarele efecte:

- umple fereastra cu blancuri (curăță fereastra);
- atribuie întregii ferestre culoarea sa de fond (declarată cu TEXTBACKGROUND);
- poziționează cursorul în colțul din stânga sus.

Prin rularea programului următor vă veți convinge de acest lucru.

```
program f3;
uses crt;

begin
   window (10,10,20,20);
   textbackground(3);
   clrscr;
end.
```

Fixarea culorii cu care se scriu caracterele se face utilizând procedura **TEXTCOLOR**. Forma generală este:

TEXTCOLOR(culoare)

Parametrul culoare trebuie să conțină un număr între 0 și 15 sau un cuvânt ce exprimă această culoare (vezi tabelul).

După rularea programului f4 observăm următoarele:

- fereastra nu a devenit vizibilă;
- cuvântul 'text' este scris pe fondul cerut și cu culoarea solicitată.

```
program f4;
uses crt;
```

```
begin
   window (10,10,20,20);
   textbackground(3);
   textcolor(0);
   write('text')
end.
```

Pentru a face vizibilă întreaga fereastră se utilizează procedura CLRSCR.

```
program f5;
uses crt;

begin
   window (10,10,20,20);
   textbackground(3);
   textcolor(0);
   clrscr;
   write('text')
end.
```

Există posibilitatea ca fiecare text scris într-o fereastră să fie identificat prin propriile culori de fond și de scriere a caracterelor. Tocmai acesta este motivul pentru care, după execuția procedurii TEXTBACKGROUND, întreaga fereastră nu capătă culoarea de fond solicitată.

```
program f6;
uses crt;

begin
   window (10,10,20,20);
   textbackground(3);
   textcolor(0);
   clrscr;
   writeln('text1');
   textbackground(0);
   textcolor(15);
   writeln('text2')
end.
```

Nu există o procedură prin care se închide o fereastră. Prin deschiderea altei ferestre, fereastra anterioară încetează de a mai fi activă (se închide). Pentru a o activa se folosesc din nou procedurile WINDOW, TEXTBACKGROUND, TEXTCOLOR cu aceiași parametri efectivi. În exemplul care urmează se deschide o fereastră în care se citește o variabilă, se deschide o altă fereastră în care acea variabilă se tipărește, apoi se redeschide prima fereastră în care se tipărește variabila citită.

```
program f7;
uses crt;
var a:integer;

begin
   clrscr;
   window (1,1,10,10);
   textbackground(3);
   textcolor(15);
```

```
    clrscr;
    write('a=');
    readln(a);
    window(11,11,40,25);
    textbackground(2);
    textcolor(7);
    clrscr;
    write('a=',a);
    window(1,1,10,10);
    textbackground(3);
    textcolor(15);
    clrscr;
    write('a=',a)
end.
```

În finalul acestui paragraf facem o observație: toate procedurile prezentate aici (ale unității CRT) lucrează asupra memoriei video. Acest lucru îl putem face și noi, fără ajutorul lor, însă să recunoaștem că este mai ușor să le folosim pe cele gata făcute.

7.3. Alte proceduri și funcții specifice unității CRT

Până acum, în cazul tuturor scrierilor și citirilor efectuate nu aveam un control asupra poziției cursorului. Din acest motiv, rezultatele programelor noastre nu erau estetice (obțineam rezultate corecte, însă scrise pe un ecran care conținea și lucruri care nu interesează în cadrul aplicației).

Procedura **GOTOXY** ne scapă de astfel de inconveniente. Ea are rolul de a poziționa cursorul într-o anumită poziție.

Forma generală este:

GOTOXY(x,y),

unde x și y reprezintă coloana și linia de poziționare a cursorului.

În programul următor se citește o valoare numerică. În situația în care, din greșeală, se introduce o valoare nenumerică, aceasta este ștearsă și se face o nouă citire. Nouă este modalitatea de citire într-o poziție unică. Tot aici, observăm o nouă procedură a CRT numită **CLREOL**. Forma generală a acestei proceduri este:

CLREOL

și are rolul de a șterge linia pe care se află cursorul, începând din poziția curentă a acestuia.

```
program af1;
uses crt;
var a:integer;

begin
   clrscr;
   gotoxy(10,10);
   write('dati o valoare numerica ');
   repeat
      gotoxy(35,10);
```

```
        clreol;
        {$i-} read(a) {$i+}
        until ioresult=0
end.
```

Procedura **DELLINE** șterge întreaga linie pe care se află cursorul.

Funcțiile numerice **WHEREX, WHEREY** (fără parametri formali) întorc coloana respectiv linia pe care se află cursorul.

În programul af2 se scriu trei mesaje pe rânduri diferite (saltul de la un rând la altul se face cu GOTOXY) și rândul din mijloc este șters. Cursorul se poziționează pe rândul următor.

```
program af2;
uses crt;

begin
   clrscr;
   write('am scris un mesaj');
   gotoxy(1,wherey+1);
   write('scriu alt mesaj');
   gotoxy(1,wherey+1);
   write('al treilea mesaj');
   gotoxy(1,wherey-1);
   delline
end.
```

Pentru citiri speciale de la tastatură, unitatea CRT conține două funcții READKEY și KEYPRESSED.

Funcția **READKEY** este de tip caracter fără parametri formali și întoarce caracterul tastat. Dacă în timpul execuției acestei funcții nu se apasă nici o tastă, programul așteaptă. Citirea caracterului se face fără ecou (adică acel caracter nu apare pe ecran).

Este bine să rulăm programul următor, spre a vedea „pe viu" execuția acestei funcții.

```
program af3;
uses crt;
var c:char;

begin
   c:=readkey
end.
```

Utilizată în mod inteligent, această funcție se dovedește a fi extrem de utilă. Multe programe rămân în așteptare până se apasă o anumită tastă. Unele taste returnează caractere cu codul cuprins într-un octet, altele (speciale) returnează caractere cu codul pe doi octeți (din care primul conține numărul 0). Săgețile (extrem de folosite pentru realizarea unor programe cu interfața prietenoasă) intră în cea de-a doua categorie. În cazul caracterelor cu codul pe doi octeți, funcția returnează numai primul octet, dar reapelată, îl returnează și pe cel de-al doilea. Astfel, dacă la primul apel codul este 0, reapelăm funcția. Citirea caracterelor fără ecou constituie de asemenea un mare avantaj (servește la introduceri de parole sau citirea unor taste speciale de la care așteptăm o

acțiune). În programul următor se arată cum se pot citi tastele corespunzătoare săgeților în sine. Programul nu realizează mare lucru, dar mecanismul rămâne valabil pentru secvențe mai interesante.

```
program af4;
uses crt;
var c:char;

begin c:=readkey;
   if c=#0 then c:=readkey;
   case c of
      #72: write ('am apasat sageata sus');
      #80: write ('am apasat sageata jos');
      #75: write ('am apasat sageata stânga');
      #77: write ('am apasat sageata dreapta')
   else write('nu am apÐsat sageti')
   end {case}
end.
```

Funcția booleană **KEYPRESSED**, fără parametri formali, returnează valoarea TRUE dacă a fost apăsată o tastă, în caz contrar valoarea returnată fiind FALSE. Este folosită în situații în care se rulează o secvență până când este apăsată o tastă. Programul următor așteaptă apăsarea unei taste pentru a se opri.

```
program af5;
uses crt;

begin
   repeat
   until keypressed
end.
```

Observație: în situația în care această funcție se reapelează trebuie golit bufferul (utilizând READLN fără parametri), altfel funcția returnează valoarea TRUE, fără ca vreo tastă să fie apăsată. De asemenea, există taste care, chiar apăsate, nu au nici un efect asupra acestei funcții (ALT, SHIFT etc.).

De multe ori, pentru avertizarea operatorului (sau în jocuri) programul trebuie să emită sunete. Pentru aceasta, în unitatea CRT se găsesc două proceduri SOUND și NOSOUND.

Procedura **SOUND** este de forma:

SOUND (h:Word).

Parametrul h furnizează frecvența sunetelor (în Hz). După apelul acestei proceduri sunetul persistă până când se apelează procedura **NOSOUND**. Această procedură nu are parametri și rolul ei este de a opri sunetul lansat prin SOUND. Procedura **DELAY** pune programul în așteptare pentru o perioadă de timp. Forma generală a acesteia este:

DELAY(n:word).

Parametrul n exprimă timpul de așteptare exprimat în milisecunde.

Combinate, cele trei proceduri pot produce un sunet o anumită durată de timp.

```
program af6;
uses crt;

begin
   sound(1000);
   delay(2000);
   nosound
end.
```

7.4. Aplicații

7.4.1. Unitatea de program UTIL

În paragrafele anterioare s-a arătat modul de lucru cu ferestre. Pentru a scrie cu ușurință programe în care intervin aceste ferestre vom prezenta o unitate de program (numită UTIL) care conține o serie de proceduri utile în acest caz. Tot aici se găsesc definite constante, o serie de caractere din codul ASCII cu care vom lucra des. În continuare, prezentăm pe larg procedurile conținute în unitate.

7.4.1.1. Procedura DESCHID_F

Are rolul de a deschide o fereastră la care au fost precizate culorile de fond și de scriere a caracterelor. Odată deschisă, fereastra capătă culoarea de fond.

Forma generală de apel pentru această procedură este:

DESCHID_F (xc1,yc1,xc2,yc2,fd,cnl);

Toți parametrii sunt de tip byte, sunt transmiși prin valoare și au următoarea semnificație:

xc1,yc1 – coordonatele colțului din stânga sus al ferestrei;
yc1,yc2 – coordonatele colțului din dreapta jos al ferestrei;
fd,cnl – culorile de fond și de scriere a caracterelor.

Programul următor deschide o fereastră utilizând această procedură.

```
program u1;
uses util;
var a:integer;

begin
   deschid_f(3,3,20,20,rosu,verde);
   write('a=');
   readln(a);
end.
```

7.4.1.2. Procedura SCRIU_F

Are rolul de a scrie un text în fereastră. Scrierea se face direct în memoria video. În acest fel, în cazul în care se scrie în ultima linie a unei ferestre, nu mai apare efectul de **SCROLL** (toate liniile urcă și ultima linie se eliberează). În

scrierea meniurilor, acest lucru se dovedește extrem de util.

Apelul acestei proceduri se realizează astfel:

SCRIU_F(sir,xc,yc,coloana,linie)

Toți parametrii sunt transmiși prin valoare și au semnificația următoare:

șir	– un șir de tip STRING ce se va tipări;
xc,yc	– coordonatele colțului din stânga sus al ferestrei;
coloana,linie	– coloană și linia din cadrul ferestrei în care va fi scris mesajul.

În programul următor se scrie un mesaj pe ultima linie a unei ferestre ce se creează.

```
program u2;
uses util;
var mesaj:string;

begin
   deschid_f(3,3,10,20,verde,rosu);
   scriu_f('12345678',3,3,1,18);
end.
```

Este important ca lungimea mesajului să nu depășească numărul de caractere ce se pot scrie pe o linie (procedura nu efectuează acest control).

Cum s-a realizat această procedură? Dispunem de coordonatele colțului din stânga sus al ferestrei. Aceste coordonate sunt aceleași și în cazul în care ne adresăm direct memoriei video. Punctul de coordonate (coloană,linie) în cadrul ferestrei va avea coordonatele (xc+coloana−1,yc+coloana−1) în cadrul ecranului. Astfel, punctul de coordonate (1,1) în cadrul ferestrei va avea coordonatele (xc,yc) în ecranul complet, punctul din coloana 2 și linia 3 va avea coordonatele absolute (xc+1,yc+2). Începând din punctul ales, după conversia coordonatelor, șirul se copiază caracter cu caracter pe linie.

Următoarele proceduri ce vor fi prezentate se vor dovedi utile atunci când scriem rutine pentru meniuri. În ce constau acestea?

Unele programe pot efectua mai multe operații, după dorința utilizatorului. Există forme primitive de a solicita opțiunea (de exemplu o cifră) sau forme evoluate (prin **meniuri**). În acest din urmă caz, programul afișează un tablou dreptunghiular în care sunt scrise opțiunile. Opțiunea luată în considerație este scrisă în video invers (vom vedea o bară luminoasă deasupra ei). Cu ajutorul săgeților se poate lua în considerație orice opțiune (prin deplasarea acelei bare luminoase). În momentul în care tastăm ENTER, programul execută opțiunea pe care este plasată bara luminoasă. Dacă se tastează ESC, înseamnă că nu ne-am hotărât asupra vreunei opțiuni. În acest caz fereastra se închide.

7.4.1.3. Procedura A_D_BARA

Are un dublu rol:

- în situația în care se dorește apariția barei luminoase deasupra unei opțiuni;
- în situația în care bara este prezentă și se dorește ca ea să dispară.

Modalitatea de apel este următoarea:

A_D_BARA(xc,yc,coloana,linia,lungime).

Toți parametri formali sunt de tip byte și au semnificația:

(xc,yc) – coordonatele colțului din stânga sus al ferestrei de meniu;
(coloana,linia) – coordonatele primului caracter căruia i se vor inversa atributele de culoare;
lungime – numărul de caractere pentru care se inversează atributele de culoare (lungimea barei).

Care este principiul de lucru al acestei proceduri? Apariția barei luminoase se face prin inversarea culorilor de fond și de scriere a caracterelor (ne aducem aminte că pentru fiecare caracter ce apare pe ecran există în memoria video un octet care reține atributele de culoare) iar dacă bara este prezentă, reinversarea culorilor va duce la dispariția ei.

Pentru a realiza inversarea culorilor se procedează astfel:

- se identifică primul caracter pentru care trebuie reiversate culorile (mecanismul de identificare este identic cu cel folosit în procedura SCRIU_F);
- din informația conținută de octetul de atribute al acestui caracter se identifică cele două culori:
  - culoarea de scriere (ultimii 4 biți ai octetului) reprezintă restul împărțirii la 16 a numărului conținut în octet;
  - culoarea de fond (de la bitul al doilea până la bitul al patrulea) se obține izolând conținutul primilor 4 biți (câtul întreg din împărțirea la 16) și prin izolarea din cadrul acestuia a informației conținute în ultimii 3 biți (restul împărțirii la 16);
- se construiește noul octet de atribute prin înmulțirea cu 16 a numărului care reprezenta culoarea la care se adaugă numărul care reprezintă fondul (în acest fel numărul ce reprezenta culoarea va ocupa primele 4 poziții din octet și numărul ce reprezenta fondul ultimele poziții);
- operațiile descrise se fac pentru toate caracterele ce se află pe lungimea barei, începând cu primul.

Observație: pentru a reuși inversarea, pentru ambele culori se folosesc numai coduri între 0 și 7.

7.4.1.4. Procedura PLIMB_BARA_V

Are rolul de a deplasa bara luminoasă asupra meniului de sus în jos și invers cu ajutorul săgeților (sus, jos). În situația în care se dă comanda sus și bara se află plasată pe prima poziție (de sus), ea va apărea în ultima poziție. În situația în care bara se află plasată pe ultima poziție și se dă comanda jos, ea va apărea pe prima poziție (bara se deplasează circular). Această procedură se activează numai în situația în care bara luminoasă este prezentă deasupra unei opțiuni (inițial se activează A_D_BARA pe prima opțiune). Procedura returnează trei valori și anume:

- (xb,yb) coordonatele unde a rămas bara atunci când s-a tastat ENTER sau ESC;
- codul ENTER sau ESC.

Cu aceste informații se decide în programul care utilizează aceste proceduri ce este de făcut și anume:

- dacă s-a tastat ESC fereastra se închide;
- dacă s-a tastat ENTER, se analizează coordonatele xb, yb spre a vedea care anume opțiune a fost cerută pentru ca programul să o execute.

Analiza modului de realizare a acestei proceduri ramine în seama cititorului. Apelul se realizează prin:

PLIMB_BARA_V(x1,y1,x2,y2,lung,xb,yb,c);

Toți parametrii formali sunt de tip byte. Ultimii trei parametri sunt transmiși prin referință (se declară cu **var** în blocul apelator), restul sunt transmiși prin valoare. Semnificația lor este următoarea:

(x1,y1) – coordonatele colțului din stânga sus al ferestrei;
(x2,y2) – coordonatele colțului din dreapta jos al ferestrei;
lung – lungimea barei;
xb,yb – coordonatele primului caracter al barei (se transmit la apel coordonatele inițiale și se regăsesc la revenirea în program coordonatele primului punct în care a rămas bara);
c – codul caracterului cu care s-a revenit în blocul apelator (poate fi ENTER sau ESC).

7.4.1.6. Procedura PLIMB_BARA_O

Are rolul de a plimba bara pe direcție orizontală (se utilizează în cadrul meniurilor orizontale). Se apelează prin:

PLIMB_BARA_O(x1,y1,x2,y2,lung,p,xb,yb,c);

x1,y1,x2,y2,lung,xb,yb,c – au aceeași semnificație și sunt de același tip cu cei prezentați la procedura anterioară;
p – se transmite prin referință, este de tip byte și permite ca între două apariții cosecutive ale barei să se sară un număr de caractere (acest număr trebuie încărcat în p la apelul procedurii).

7.4.1.6. Procedura CURSOR

Are rolul de a face să apară sau să dispară cursorul (este lipsit de eleganță ca în cadrul unui meniu cursorul să fie prezent).

Se apelează prin

CURSOR(TRUE) sau CURSOR(FALSE).

Primul caz duce la apariția cursorului iar în cel de-al doilea caz – la dispariția lui. Modul de realizare a acestei proceduri depășește cadrul acestui manual. Procedura este prezentată pentru utilitatea ei.

În partea de implementare a unității de program se găsește și funcția de tip byte **CODUL**. Această funcție are rolul de a citi codul unei taste apăsate și este utilizată de celelalte proceduri ale unității. La prezentarea procedurii READKEY s-a explicat modul în care se citește codul unei taste apăsate; această procedură nu face altceva decât să concretizeze cele deja arătate.

În continuare, vom scrie un program care folosește un meniu simplu de tip vertical.

```
program u3;
uses util,crt;
var xb,yb,c:byte;

begin
   deschid_f(50,20,53,23,rosu,negru);
   scriu_f('opt1',50,20,1,1);
   scriu_f('opt2',50,20,1,2);
   scriu_f('opt3',50,20,1,3);
   scriu_f('opt4',50,20,1,4);
   cursor(false);
   a_d_bara(50,20,1,1,4);
   xb:=1; yb:=1;
   plimb_bara_v(50,20,53,23,4,xb,yb,c);
   deschid_f(1,1,80,25,negru,alb);
   cursor(true);
   gotoxy(1,1);
   if c=enter then
      case yb of
         1: writeln('optiunea 1');
         2: writeln('optiunea 2');
         3: writeln('optiunea 3');
         4: writeln('optiunea 4')
      end {case}
end.
```

Se deschide o fereastră cu 4 linii și 4 coloane. Se va scrie cu negru pe un fond roșu. Procedăm astfel:

- se scriu cele 4 opțiuni posibile cu ajutorul procedurii SCRIU_F:
- se elimină apariția cursorului prin apelul procedurii CURSOR;
- se activează bara luminoasă pe prima linie a meniului (A_D_BARA);
- se activează procedura PLIMB_BARA_V pe prima linie și coloană a ferestrei;
- în cazul în care s-a tastat ENTER se deschide o altă fereastră ce marchează întreg ecranul și se scrie ce opțiune a fost selectată (un program complex, în acest punct, ar fi activat subprogramele care efectuau operația selectată de utilizator).

Iată unitatea de program ce a fost prezentată:

```
unit util;
interface
const
sus       =72;
jos       =80;
dreapta   =77;
stânga    =75;
enter     =13;
esc       =27;
colt_s_s    = #201;
colt_d_s    = #187;
colt_d_j    = #188;
colt_s_j    = #200;
orizontal   = #205;
vertical    = #186;
vertical1   = #179;
orizontal1  = #196;
```

```
colt_s_s1   = #218;
colt_s_j1   = #192;
colt_d_s1   = #191;
colt_d_j1   = #217;
Negru=0; Albastru=1; Verde=2; Bleu=3; Rosu=4; Mov=5; Maro=6;
GriDeschis=7; GriInchis=8; AlbastruDeschis=9; VerdeDeschis=10;
BleuDeschis=11; RosuDeschis=12; RozDeschis=13; Galben=14;
Alb=15; Clipitor=16;

procedure deschid_f(xc1,yc1,xc2,yc2,fd,cn1:byte);
procedure scriu_f(sir:string;xc,yc,coloana,linie:byte);
procedure a_d_bara(xc,yc,coloana,linie,lung:byte);
procedure plimb_bara_v(x1,y1,x2,y2,lung:byte;var xb,yb,c:byte);
procedure plimb_bara_o(x1,y1,x2,y2,lung,p:byte;var
   xb,yb,c:byte);
procedure cursor(afisare:boolean);

implementation

uses CRT,DOS;
type aleg=(caracter,atribute);
var ecran :array[1..25,1..80,caracter..atribute] of char
        absolute $b800:0;
    ecran1 :array[1..25,1..80,caracter..atribute] of byte
        absolute $b800:0;

procedure deschid_f;
begin
   Window(xc1,yc1,xc2,yc2);
   TextBackground(fd);
   TextColor(cn1);
   clrscr
end;

procedure scriu_f;
var x,y,i:byte;
begin
   x:=xc+coloana-1;
   y:=yc+linie-1;
   for i:=0 to length(sir)-1 do ecran[y,x+i,caracter]:=sir[i+1];
end;

procedure a_d_bara;
var
   i,x,y,cn1,fd:integer;
begin
   x:=xc+coloana-1;
   y:=yc+linie-1;
   cn1:=ecran1[y,x,atribute] mod 16;
   fd:=(ecran1[y,x,atribute] div 16) mod 8;
   for i:=0 to lung-1 do
      ecran1[y,x+i,atribute]:=cn1*16+fd;
end;

function codu1:byte;
var c:char;
begin
   c:=readkey;
   if ord(c)=0 then c:=readkey;
   codu1:=ord(c)
```

```
end;
procedure plimb_bara_v;
begin
   repeat
      c:=codul;
      case c of
      sus:  if yb=1 then
            begin
               a_d_bara(x1,y1,xb,yb,lung);
               yb:=y2-y1+1;
               a_d_bara(x1,y1,xb,yb,lung)
            end
            else
            begin
               a_d_bara(x1,y1,xb,yb,lung);
               yb:=yb-1;
               a_d_bara(x1,y1,xb,yb,lung)
            end;
      jos:  if yb=y2-y1+1  then
            begin
               a_d_bara(x1,y1,xb,yb,lung);
               yb:=1;
               a_d_bara(x1,y1,xb,yb,lung)
            end
            else
            begin
               a_d_bara(x1,y1,xb,yb,lung);
               yb:=yb+1;
               a_d_bara(x1,y1,xb,yb,lung)
            end;
      end; {case}
   until (c=esc) or (c=enter);
end;
procedure plimb_bara_o;
begin
   repeat
   c:=codul;
   case c of
      stânga:  if xb=1 then
               begin
                  a_d_bara(x1,y1,xb,yb,lung);
                  xb:=x2-x1-lung+2;
                  a_d_bara(x1,y1,xb,yb,lung)
               end
               else
               begin
                  a_d_bara(x1,y1,xb,yb,lung);
                  xb:=xb-lung-p;
                  a_d_bara(x1,y1,xb,yb,lung)
               end;
      dreapta:    if xb+lung-2 = x2-x1 then
                  begin
                     a_d_bara(x1,y1,xb,yb,lung);
                     xb:=1;
                     a_d_bara(x1,y1,xb,yb,lung)
                  end
                  else
```

```
                    begin
                       a_d_bara(x1,y1,xb,yb,lung);
                       xb:=xb+lung+p;
                       a_d_bara(x1,y1,xb,yb,lung)
                    end;
      end; {case}
   until (c=esc) or (c=enter);
end;

procedure cursor;
var r:registers;
begin
   r.ah:=1;
   if afisare then
   begin
      r.ch:=6;
      r.cl:=7;
   end
   else
   begin
      r.ch:=32;
      r.cl:=7
   end;
   intr(16,r);
end;
end.
```

7.4.2. Fereastra obiect

Până acum am învățat să construim o fereastră. De asemenea, avem și proceduri care ușurează mult programarea în cazul lucrului cu ferestre. De ce nu am avea și o fereastră obiect? În acest mod, o fereastră va fi pentru programul nostru o variabilă.

Ca orice obiect, obiectul **FEREASTRA** va conține date și metode. **Datele** sunt următoarele:

- nr_f reprezintă numărul ferestrei;
- xcs,ycs,xcj,ycj sunt coordonatele colțurilor din stânga sus și dreapta jos ale ferestrei;
- fond,culoare – culorile de fond și de scriere a caracterelor;
- tip_f – reprezintă tipul ferestrei (0 – fereastră fără chenar, 1 – fereastră cu chenar simplu, 2 – fereastră cu chenar dublu);
- xcursor,ycursor – pozițiile curente ale cursorului;
- vizibil – variabila booleană care reține dacă este vizibil sau nu cursorul.

Metodele conținute de obiectul FEREASTRA sunt următoarele:

7.4.2.1. Metoda DESCHID

Se apelează astfel:

nume variabilă.DESCHID(x1,y1,x2,y2,fd,cl,n,tip,vizibil);

Parametrii au următoarea semnificație:

- x1,y1,x2,y2: coordonatele colțurilor din stânga sus și dreapta jos ale

ferestrei;
- fd,cl – culorile de fond și cerneala;
- n – numărul ferestrei;
- tip – tipul ferestrei (fără chenar – 0, chenar simplu – 1,chenar dublu – 2);
- vizibil – parametru ce precizează dacă este vizibil sau nu cursorul în fereastră.

Cu excepția ultimului parametru (care este de tip boolean) restul sunt de tip byte. Toți parametrii sunt transmiși prin valoare.

7.4.2.2. Metoda SALVEZ

Are rolul de a salva conținutul unei ferestre (salvarea se face în memoria internă a calculatorului; de fapt se salvează întregul ecran pentru a nu pierde și contextul în care se găsește fereastra). Se apelează prin:

nume variabila.SALVEZ

7.4.2.3. Metoda RESTAUREZ

Are rolul de a reface o fereastră pe ecran (o aduce din locul de unde a fost salvată). Apelul se face prin:

nume variabilă.RESTAUREZ.

Mecanismul de realizare al ultimelor două proceduri îl vom învăța anul următor.

Programul următor exemplifică lucrul cu acest obiect și realizează:
- se declară două variabile de tip fereastră;
- pentru întreg ecranul se stabilește fondul negru și culoarea de scriere a caracterelor – albă.
- se deschide o fereastră pe care se scrie o linie (fereastra are chenar simplu iar cursorul este prezent);
- această fereastră se salvează;
- se deschide o altă fereastră cu chenar dublu în care cursorul nu este prezent;
- se scrie prima linie a acestei ferestre;
- se restaurează prima fereastră.

```
program fer1;
uses fers,util,crt;
var a,b:fereastra;

begin
   textbackground(negru);
   textcolor(alb);
   clrscr;
   a.deschid(1,1,5,5,negru,rosu,1,1,true);
   write(1,2,3);
   delay(3000);
   a.salvez;
   b.deschid(1,1,10,10,rosu,negru,2,2,false);
   write(1,2,3,4,5,6,7,8);
   delay(3000);
   a.restaurez;
end.
```

În continuare se prezintă unitatea ce conține obiectul FEREASTRA.

```
unit fers;
interface
type  ecran=array[1..25,1..80,1..2] of byte;
type  ref=^ecran;
      fereastra=object
                  nr_f,xcs,ycs,xcj,ycj,fond,tip_f,
                  culoare,xcursor,ycursor:byte;
                  viz:boolean;
                  procedure deschid (x1,y1,x2,y2,fd,cl,n,
                     tip:byte;vizibil:boolean);
                  procedure salvez;
                  procedure restaurez;
                  end;

implementation

uses util,crt;
var v:array[1..9] of ref;

procedure fereastra.deschid;
var i:integer;
begin
   nr_f:=n;
   tip_f:=tip;
   xcs:=x1;
   ycs:=y1;
   xcj:=x2;
   ycj:=y2;
   fond:=fd;
   culoare:=cl;
   viz:=vizibil;
   cursor(vizibil);
   deschid_f(x1,y1,x2,y2,fd,cl);
   if tip <>0 then
   begin
      if tip=1    then write(colt_s_s1)
                  else write(colt_s_s);
      for i:=2 to xcj-xcs do
      begin
         gotoxy(i,1);
         if tip=1    then write(orizontal1)
                     else write(orizontal);
         gotoxy(i,ycj-ycs+1);
         if tip=1    then write(orizontal1)
                     else write(orizontal);
      end;
      gotoxy(xcj-xcs+1,1);
      if tip=1    then write(colt_d_s1)
                  else write(colt_d_s);
      for i:=2 to ycj-ycs do
      begin
         gotoxy(1,i);
         if tip=1    then write(vertical1)
                     else write(vertical);
         gotoxy(xcj-xcs+1,i);
         if tip=1    then write(vertical1)
```

```
                    else write(vertical);
      end;
      gotoxy(1,ycj-ycs+1);
      if tip=1    then write(colt_s_j1)
                  else write(colt_s_j);
      if tip=1 then scriu_f(colt_d_j1,xcs,ycs,xcj-xcs+1,ycj-ycs+1)
               else scriu_f(colt_d_j,xcs,ycs,xcj-xcs+1,ycj-ycs+1);
      deschid_f(xcs+1,ycs+1,xcj-1,ycj-1,3,7);
   end;
end;

procedure fereastra.salvez;
var b:ecran absolute $b800:0;
begin
   new(v[nr_f]);
   v[nr_f]^:=b;
   xcursor:=wherex;
   ycursor:=wherey;
end;

procedure fereastra.restaurez;
var b:ecran absolute $b800:0;
begin
   b:=v[nr_f]^;
   dispose(v[nr_f]);
   window(xcs+1,ycs+1,xcj-1,ycj-1);
   gotoxy(xcursor,ycursor);
   cursor(viz);
end;
end.
```

7.4.3. Meniul obiect

Pentru scrierea cu ușurință a meniurilor, în cele ce urmează se prezintă obiectul **MEN** (obiect de tip meniu) care se găsește plasat în unitatea MENU. Acest obiect moștenește obiectul FEREASTRA (deci toate datele și metodele prezentate în cadrul acelui obiect se regăsesc aici). În plus, acesta are alte date și metode.

Datele noi sunt:

- xb, yb: coordonatele în cadrul ferestrei ale primului caracter al barei luminoase;
- lung, hi: lungimea și înălțimea ferestrei de meniu;
- răsp: răspunsul operatorului în cadrul selecției (ENTER sau ESC);
- lbar: lungimea barei luminoase.

Prezentăm noile **metode** ale obiectului MEN.

Metoda **DESCHID_M** are următoarele funcții:

- construiește meniul;
- activează bara luminoasă pe prima opțiune a acestuia;
- permite operatorului să manipuleze bara asupra opțiunilor și întoarce răspunsul acestuia (selecție și ce anume s-a selectat (ENTER) sau lipsa selecției (ESC)).

Dacă înălțimea ferestrei de meniu este 1, meniul se consideră orizontal, în

caz contrar meniul este vertical. Apelul metodei se realizează astfel:

nume variabilă.deschidm(x1,y1,l,h,fd,cl,n,lb,a,vizibil);

Toți parametrii se transmit prin valoare. Primii 8 parametri sunt de tip byte, parametrul a este de tip string, parametrul vizibil este boolean. Semnificația parametrilor este următoarea:

- x1, y1 – coordonatele colțului din stânga sus al ferestrei meniu;
- l, h – lungimea și înălțimea ferestrei meniu;
- fd,cl – culorile de fond și de scriere a caracterelor;
- n – numărul ferestrei meniu;
- lb – lungimea barei luminoase;
- vizibil – parametru ce precizează dacă se vede sau nu cursorul.

Precizăm faptul că, în cazul unui meniu vertical, șirul de tip string ce conține opțiunile se împarte automat într-un număr de părți egale ca lungime. Numărul de părți este dat de numărul de linii din meniu.

Am învățat faptul că nu este permis să se acceseze datele din cadrul unei variabile obiect decât prin intermediul metodelor pe care le are obiectul. Astfel am înzestrat obiectul cu următoarele metode de tip funcție:

- SELECTAT – funcție booleană ce specifică dacă revenirea din selecție s-a făcut prin ENTER:
- ÎNCHIS – funcție booleană ce specifică dacă revenirea din selecție s-a făcut prin ESC (de fapt ar putea exista o singură funcție, întrucât dintr-un meniu nu se poate reveni decât cu ENTER sau ESC);
- funcțiile CITXB, CITYB de tip byte returnează poziția barei în momentul selecției (coloană, linie).

Obiectul MEN mai are o metodă de tip procedură fără parametri numită RESTAUREZM. Aceasta are rolul ca, pe lângă restaurarea unui meniu salvat anterior, să activeze și bara luminoasă (se folosește în cadrul meniurilor duble (lansez un meniu din cadrul altui meniu).

În programul următor se realizează un meniu orizontal.

```
program m1;
uses menu,util,crt;
var a:men;
begin
   textbackground(negru);
   clrscr;
   a.deschidm(20,10,9,1,verde,rosu,1,4,'opt1 opt2',false);
   if a.selectat then
   begin
      deschid_f(1,1,80,25,negru,alb);
      if a.citxb=1  then writeln('optiunea 1')
                    else writeln('optiunea 2')
   end
end.
```

În programul m2 se realizează un meniu vertical.

```
program m2;
uses menu,crt,util;
var a:men;
```

```
begin
   textbackground(negru);
   clrscr;
   a.deschidm(10,10,4,4,verde,rosu,1,4,'opt1opt2opt3opt4',false);
   if a.selectat then
   begin
      deschid_f(1,1,80,25,negru,alb);
      case a.cityb of
         1: writeln('optiunea 1');
         2: writeln('optiunea 2');
         3: writeln('optiunea 3');
         4: writeln('optiunea 4')
      end {case}
   end
end.
```

În programul care urmează se construiește un meniu dublu. Inițial avem un meniu orizontal cu două opțiuni (adunare / scădere sau înmulțire / împărțire). Odată ce am selectat o astfel de opțiune se deschide un alt meniu vertical ce prezintă alte două opțiuni. De exemplu, dacă în primul meniu am selectat ADUNARE / SCĂDERE acum trebuie să ne pronunțăm dacă dorim ADUNARE sau SCĂDERE. Oricare meniu se poate închide prin utilizarea tastei ESC, caz în care se revine în meniul anterior sau se termină programul (în situația în care tastăm ESC în cadrul meniului orizontal). Indiferent de operația solicitată, se deschide o fereastră în care aceasta este executată.

```
program f5;
uses menu,crt,util,fers;
var   a,b,c:men;
      d,e,f:string;
      g:fereastra;
      m,n:real;

procedure adunare;
begin
   write ('m='); readln(m);
   write ('n='); readln(n);
   write('s=',m+n:3:2);
   delay(1000)
end;

procedure scadere;
begin
   write ('m='); readln(m);
   write ('n='); readln(n);
   write('s=',m-n:3:2);
   delay(1000)
end;

procedure înmultire;
begin
   write ('m='); readln(m);
   write ('n='); readln(n);
   write('p=',m*n:3:2);
   delay(1000)
end;

procedure împartire;
begin
```

```
   write ('m='); readln(m);
   write ('n='); readln(n);
   if n<>0  then write('c=',m/n:3:2)
            else write('num=0');
   delay(1000)
end;

begin
   textbackground(0);
   clrscr;
   d:='ADUN/SCAD ÎNML/ÎMPR';
   e:='ADUNARE SCADERE ';
   f:='ÎNMULTIRE  ÎMPARTIRE ';
   a.deschidm(20,1,20,1,Bleu,GriDeschis,1,10,d,false);
   repeat
      if a.selectat then
         if a.citxb=1 then
         begin
            a.salvez;
            b.deschidm(10,3,8,2,Bleu,GriDeschis,2,8,e,false);
            repeat
               if b.selectat then
                  if b.cityb=1 then
                  begin
                     b.salvez;
                     g.deschid(10,3,20,20,Albastru,Rosu,4,2,true);
                     adunare;
                     b.restaurezm
                  end
                  else
                  begin
                     b.salvez;
                     g.deschid(10,3,20,20,Albastru,Rosu,4,2,true);
                     scadere;
                     b.restaurezm
                  end
            until b.inchis
         end
         else
         begin
            a.salvez;
            c.deschidm(10,3,10,2,Bleu,GriDeschis,3,10,f,false);
            repeat
               if c.selectat then
                  if c.cityb=1 then
                  begin
                     c.salvez;
                     g.deschid(10,3,20,20,Albastru,Rosu,4,2,true);
                     înmultire;
                     c.restaurezm
                  end
                  else
                  begin
                     c.salvez;
                     g.deschid(10,3,20,20,Albastru,Rosu,4,2,true);
                     împartire;
                     c.restaurezm
                  end
            until c.închis
```

```
        end;
      if a.selectat then restaurezm
   until a.inchis
end.
```

În final, prezentăm unitatea de program ce conține obiectul MEN.

```
unit menu;

interface

uses fers,util;
type men=  object(fereastra)
              xb,yb,rasp,lung,hi,lbar:byte;
              procedure deschidm(x1,y1,l,h,fd,cl,n,lb:byte;
                                 a:string; vizibil:boolean);
              procedure restaurezm;
              function selectat:boolean;
              function citxb:byte;
              function cityb:byte;
              function închis:boolean;
           end;

implementation

procedure men.deschidm;
var   i:byte;
      c:string;
begin
   lung:=1;hi:=h;lbar:=lb;
   deschid(x1,y1,x1+l-1,y1+h-1,fd,cl,n,0,vizibil);
   cursor(false);
   if h=1 then
   begin
      scriu_f(a,x1,y1,1,1);
      a_d_bara(x1,y1,1,1,lb);
      xb:=1;yb:=1;
      plimb_bara_o(x1,y1,x1+l-1,y1+h-1,lb,0,xb,yb,rasp);
   end
   else
   begin
      for i:=1 to length(a) div l do
      begin
         c:=copy(a,(i-1)*l+1,l);
         scriu_f(c,x1,y1,1,i);
      end;
      a_d_bara(x1,y1,1,1,l);
      xb:=1;
      yb:=1;
      plimb_bara_v(x1,y1,x1+l-1,y1+h-1,l,xb,yb,rasp)
   end;
end;

function men.selectat:boolean;
begin
   selectat:= (rasp=enter)
end;

function men.citxb:byte;
begin
```

```
   citxb:=xb
end;

function men.cityb:byte;
begin
   cityb:=yb
end;

function men.închis:boolean;
begin
   închis:= (rasp=esc);
end;

procedure men.restaurezm;
begin
   restaurez;
   if hi<>1 then
      plimb_bara_v(xcs,ycs,xcs+lung-1,ycs+hi-1,lung,xb,yb,rasp)
   else
   plimb_bara_o(xcs,ycs,xcs+lung-1,ycs+hi-1,lbar,0,xb,yb,rasp);
end;
end.
```

7.5. Aplicații la capitolul 7

7.5.1. Aplicații ale unității de program CRT

1. Să se scrie o procedură care citește o parolă (citirea unei parole se face fără să se vadă pe ecran ce se tastează).

2. Să se scrie o procedură care citește o valoare numerică pe care o validează (din punct de vedere al naturii numerice), iar în caz că nu este validă, calculatorul va emite un sunet o perioadă, după care se va relua citirea. Citirea se face în poziție fixă.

3. Să se scrie o procedură care citește o valoare de tip STRING. Valoarea citită se validează din punct de vedere al naturii alfabetice, iar în cazul în care nu corespunde se va emite un sunet, după care citirea se va relua în același loc.

4. Să se scrie o procedură care citește o valoare de tip STRING. În cazul în care se introduce șirul vid, citirea se va relua din aceeași poziție.

5. Să se scrie o procedură care citește o matrice de trei linii și trei coloane. Citirea se va face într-o fereastră care are o culoare distinctă de restul ecranului. Elementele se introduc în poziția curentă a cursorului (acesta va fi poziționat în așa fel încât elementele citite să fie poziționate în fereastră ca și în matrice).

6. Să se scrie o procedură care tipărește o matrice cu trei linii și trei coloane. Tipărirea se face într-o fereastră cu o culoare de fond diferită de restul ecranului. În cadrul ferestrei elementele vor fi poziționate întocmai ca într-o matrice.

7. Scrieți un program care folosește un meniu cuprinzând patru opțiuni:
- citirea unui vector;
- tipărirea unui vector;
- tipărirea sumei elementelor sale;
- ieșirea din aplicație.

8. Să se scrie un meniu cu următoarele opțiuni:
- citirea unei matrice;
- tipărirea unei matrice;
- calcule;
- ieșire aplicație.

În cazul opțiunii *calcule* se va deschide un nou meniu cu opțiunile:
- suma elementelor matricei;
- suma modulelor elementelor matricei;
- calculul elementului maxim;
- calculul elementului minim;
- revenirea în meniul principal.

La selecția unei opțiuni se va apela o procedură specifică ce rezolvă problema.

9. Să se scrie o procedură care modifică o variabilă de tip string de o lungime prestabilită în felul următor:
- modificarea se face utilizând o fereastră cu fond și culoare de desen prestabilite, în care se scrie conținutul variabilei;
- există posibilitatea plimbării cursorului în cadrul ferestrei cu ajutorul săgeților stânga și dreapta;
- există posibilitatea inserării unui caracter în stânga cursorului;
- există posibilitatea ștergerii unui caracter aflat în stânga cursorului (cu BS) sau pe poziția curentă a cursorului (cu DEL).

Observație. O astfel de procedură se dovedește extrem de utilă în două cazuri:
- cel prezentat, atunci când variabila are un anumit conținut care se modifică;
- cazul în care variabila se citește efectiv (variabila de tip string care se transmite procedurii conține numai blancuri).

10. Utilizând procedura de la problema anterioară citiți o variabilă de tip string care va fi convertită în valoare de tip INTEGER. În cazul în care conversia nu reușește, se va emite un sunet, după care se va relua citirea din aceeași poziție.

11. Citiți la fel ca la problema 10 o variabilă reală.

12. Concepeți un obiect numit CAMP care cuprinde:
- lungimea câmpului (numarul de caractere);
- o variabilă de tip string (conținutul câmpului);
- coordonatele ferestrei în care va fi tipărit câmpul pe ecran;
- culoarea de fond a ferestrei;
- culoarea cu care se scriu caracterele în această fereastră;
- o metodă care citește elegant variabila de tip string (poate fi eventual procedura de la problema 9);
- o metodă care încarcă variabila de tip string cu un șir dorit;
- o metodă care returnează valoarea de tip integer rezultată din conversia

conținutului variabilei de tip string (dacă nu reușește conversia, se va semnala);
- o metodă care returnează valoarea de tip real rezultată din conversia conținutului variabilei de tip string;
- o metodă care testează conținutul alfabetic al acestui câmp.

Observație. Odată realizat, acest obiect se va dovedi extrem de util în citirea și tipărirea diverselor valori pe care le solicită programele. Avem avantaje ca validarea simplă, posibilitatea de ștergere a unor caractere din interiorul câmpului, posibilitatea de actualizare a unor valori, etc.

13. Citiți și actualizați o variabilă de tip RECORD cu o structură la alegerea dumneavoastră. Programul va utiliza variabile obiect de tip CAMP.

Indicații / rezolvări

1. Citirea se face cu readkey. Iată procedura și un apel al ei:

```
program parola;
uses crt;
var par, par1: string;

procedure citpar(var par: string);
var c: char;
begin
   writeln('dati parola: ');
   par:='';
   repeat
      c:=readkey;
      if c=#0 then c:=readkey;
      if c<>#13 then par:=par+c
   until c=#13;
end;

begin
   par1:='Costinesti';
   citpar(par);
   if par=par1 then writeln ('ok')
end.
```

Explicați funcționarea procedurii.

2.

```
program citnum;
uses crt;
var n,i_o: integer;

procedure citn(var n:integer);
begin
   repeat
      gotoxy(10,10);
      write('n=');
      clreol;
      {$i-} read(n) {$i+};
      i_o:=ioresult;
      if i_o <>0 then
      begin
         sound(5000);
         delay(100);
         nosound
      end
   until i_o =0;
```

```
end;
begin
   citn(n);
   writeln('valoarea cititĐ este: ',n)
end.
```

Explicați funcționarea procedurii.

9. Vom prezenta procedura și un program demonstrativ care o utilizează:

```
program cite1;
uses crt, util;
var   a: string;
      i: integer;

procedure cit(var a:string; fond, culoare, x1, y1: byte);
var   c: char;
      n: byte;
begin
   n:=length(a);
   window(x1,y1,x1+n-1,y1);
   textbackground(fond);
   textcolor(culoare);
   clrscr;
   scriu_f(a,x1,y1,1,1);
   repeat
      c:=readkey;
      if c=#0 then
      begin
         c:=readkey;
         if (ord(c)=stânga) and (wherex>1) then
            gotoxy(wherex-1, wherey);
         if (ord(c)=dreapta and (wherey<n) then
            gotoxy(wherex+1, wherey);
{del}    if (c=#83) then
         begin
            a:=copy(a,1,wherex-1)+copy(a,wherex+1,-wherex)+' ';
            scriu_f(a,x1,y1,1,1);
         end;
      end
      else
      begin
         if (a[wherex]=' ') and (ord(c)<>8) and (ord(c)<>13) then
         begin
            a:=copy(a,1,wherex-1)+c+copy(a,wherex+1, n-wherex);
            scriu_f(a,x1,y1,1,1);
            gotoxy(wherex+1, wherey)
         end
         else
         if(a[wherex]<>' ')and(ord(c)<>8)and(ord(c)<>enter) then
         begin
            a:=copy(a,1,wherex-1)+c+copy(a,wherex,n-wherex);
            scriu_f(a,x1,y1,1,1);
            gotoxy(wherex+1,wherey)
         end;
{bs}     if (c=#8) and (wherex>1) then
         begin
            a:=copy(a,1,wherex-2)+copy(a,wherex,n-1)+' ';
            scriu_f(a,x1,y1,1,1);
            gotoxy(wherex-1,wherey)
         end;
      end
   until ord(c)=enter;
end;
```

```
begin
   textbackground(negru);
   textcolor(alb);
   clrscr;
   a:='12345678';
   cit(a,verde,rosu,37,10);
   window(1,1,80,25);
   textbackground(negru);
   textcolor(alb);
   clrscr;
   write(a)
end.
```

Procedura deschide o fereastră de un anumit fond și culoare în care se tipărește conținutul actual al variabilei de tip STRING (aceasta poate fi cu spații dacă se dorește numai citire). Citirea tastelor se face cu READKEY. În funcție de caracterul tastat, șirul existent în cadrul variabilei de tip string se înlocuiește cu un alt șir, după care se tipărește noul conținut al variabilei.

Se tratează următoarele cazuri:

- săgeata stânga (se deplasează cursorul la stânga, cu o poziție);
- săgeata dreapta (se deplasează cursorul spre dreapta, cu o poziție);
- DEL – se șterge caracterul aflat pe poziția curentă, la care se adaugă șirul existent în variabila de tip string până în poziția curentă, la care se adaugă șirul existent, începând cu poziția următoare celei curente până la sfârșit);
- o literă sau o cifră se tratează diferențiat astfel:
  - dacă în poziția curentă a cursorului se găsește caracterul spațiu, acesta este înlocuit cu caracterul care se tastează;
  - dacă în poziția curentă a cursorului se găsește un alt caracter, atunci noul șir se obține concatenând șirul existent până în poziția curentă – cu caracterul tastat și cu șirul care se găsește începând cu poziția curentă până la penultima poziție (se inserează un caracter);
  - dacă s-a tastat BS, noul șir se obține ștergând caracterul aflat înaintea poziției curente (ultima poziție rămâne liberă).

Atenție! Unele caractere au codul pe 2 octeți.

Capitolul 8

Fişiere Pascal

8.1. Fişiere text

Se numeşte **fişier text** o succesiune de caractere ASCII terminată cu caracterul ^Z . Putem considera fişierul ca fiind alcătuit din linii separate prin două caractere (CR şi LF) şi care nu au în mod obligatoriu lungimea egală, deşi, în practică, întâlnim de multe ori fişiere text cu o singură linie.

Fişierele text au o importanţă deosebită din două motive:
- sunt direct afişabile;
- orice editor de texte lucrează cu fişiere de acest tip.

Structura generală a unui fişier text se observă în figura de mai jos.

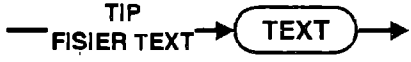


Figura 8.1.1.

Întrucât, în general, liniile nu au acelaşi număr de caractere, accesul la o anumită linie din fişier se face secvenţial (se citesc pe rând liniile, până se ajunge la linia căutată).

În program, un fişier text se declară ca variabilă de un tip predefinit şi anume **TEXT**:

VAR a,b:text;

Pentru a face legatura între numele fişierului din program şi numele său de pe suportul extern pe care se află sau se creează, se foloseşte procedura **ASSIGN**. Apelul procedurii se realizează prin:

ASSIGN (nume fişier în program, 'nume fişier pe suport extern').

Exemplul 1:

```
var a:text;
.........
assign(a,'f1.dat')
```

În acest caz, fişierul va fi creat sau se va găsi în directorul în care ne aflăm şi se numeşte f1.dat.

O formă echivalentă pentru o asignare de acest tip este:

Exemplul 2:

```
var a:text;
b:string;
.............
b:='f1.dat'
assign(a,b)
```

Exemplul 3:

Fișierul se găsește sau se va găsi pe a și va avea numele t.txt.

```
var a:text;
.............
assign (a,'a:\t.txt')
```

Exemplul 4:

Fișierul se găsește sau se va găsi pe C în subdirectorul v al directorului h și se numește util.

```
var a:text;
.............
assign(a,c:\h\v\util).
```

8.1.1. Crearea și exploatarea fișierelor text

Odată declarat și realizează legătura între numele său din program și numele său de pe suportul extern, un fișier se poate crea. Procedura care **deschide** un fișier pentru a fi creat este **REWRITE**. Forma generală este:

REWRITE (variabila fișier).

Scrierea înregistrărilor în fișier se face utilizând două proceduri — WRITE și WRITELN. Pentru început, vom face scrieri și citiri în fișier utilizând numai variabile de tip string. Din acest motiv vom prezenta o formă simplificată a procedurilor enunțate. Forma de apel simplificată a procedurii **WRITELN** este:

WRITELN (var Fișier, variabilă de tip string);

Procedura WRITELN scrie o linie în fișier, conținutul liniei este dat de conținutul variabilei de tip string și marchează sfârșitul de linie (CR,LF) .

După ce a fost creat, fișierul **se închide** (se marchează sfârșitul de fișier). Închiderea se face utilizând procedura **CLOSE**. Forma generală a acesteia este:

CLOSE (nume fișier).

Programul următor creează un fișier text cu mai multe linii.

```
program txt1;
var   f:text;
      a:string;

begin
   assign(f,'f1.dat');
   rewrite(f);
```

```
   while not eof do
   begin
      readln(a);
      writeln(f,a)
   end;
   close(f)
end.
```

Datele sunt introduse de la tastatură. Să presupunem că introducem următoarele date:

asdfghj(ENTER)
12weh(ENTER)
a g(ENTER)
1^z (ultimul caracter introdus s-a obținut tastând (CTRL+Z).

Fișierul creat va conține 4 linii:

linia 1 asdfghj
linia 2 12weh
linia 3 a g
linia 4 1

Liniile sunt separate cu CR+LF. Funcția booleană **EOF** ia valoarea TRUE dacă este întâlnit caracterul ^z (acesta va marca sfârșitul introducerii datelor).

Considerăm programul următor ce creează un fișier text.

```
program txt3;
var   f:text;
      a:string;
begin
   assign(f,'f2.dat');
   rewrite(f);
   while not eof do
   begin
      readln(a);
      write(f,a)
   end;
   close(f)
end.
```

În acest program am înlocuit procedura WRITELN cu procedura WRITE. Consecința ? La același șir de intrare, noul fișier text va avea o singură linie ce conține toate caracterele introduse, mai puțin CR. Putem acum prezenta forma simplificată a procedurii **WRITE**. Aceasta este:

WRITE (variabilă de tip string).

Are rolul de a scrie în fișier conținutul variabilei string dar nu marchează sfârșitul de linie. Astfel, la un eventual apel ulterior al acestei proceduri se va scrie în continuarea liniei.

Este normal să și putem citi fișierele create anterior.

Pentru aceasta, trebuie prezentate alte trei proceduri: RESET, READLN, READ.

Procedura **RESET** deschide un fișier pentru citire și are forma generală:

RESET (variabilă fișier);

Pentru procedurile READ, READLN se vor prezenta forme simplificate. Forma procedurii **READLN** este:

READLN (var Fișier, variabila de tip string):

Citește maximum 256 de caractere din linia curentă a fișierului. Dacă linia are mai puțin de 256 de caractere este citită toată iar conținutul efectiv al variabilei de tip string va fi șirul de caractere până la CR. Indiferent de caz, după citire se trece la linia următoare a fișierului. Funcția booleană EOF (variabilă fișier) ia valoarea TRUE dacă s-a ajuns la sfârșitul fișierului, în caz contrar ia valoarea FALSE.

Programul următor citește fișierul creat cu ajutorul lui WRITELN.

```
program txt3;
var   f:text;
      a:string;

begin
   assign(f,'f1.dat');
   reset(f);
   while not eof(f) do
   begin
      readln(f,a);
      writeln(a)
   end;
end.
```

Procedura **READ** are forma generală:

READ(var Fișier, variabila de tip string).

Pentru a înțelege exact cum funcționează READ, ne imaginăm o linie de fișier cu mai mult de 256 de caractere. După deschiderea fișierului și plasarea la începutul acestei linii apelăm pentru prima dată READ. Vor fi citite 256 de caractere. La al doilea apel vor fi citite sau următoarele 256 (dacă au mai rămas atâtea caractere necitite în linie) sau câte au mai rămas până la întâlnirea sfârșitului de linie. În programul txt4 se citește fișierul text f2.dat (cel creat pe o singură linie).

```
program txt4;
var   f:text;
      a:string;

begin
   assign(f,'f1.dat');
   reset(f);
   while not eof(f) do
   begin
      read(f,a);
      writeln(a)
   end;
end.
```

Oare variabilele de tip string sunt singurele care se pot scrie și citi din fișiere? Răspunsul la această întrebare este negativ, și acum este momentul să

prezentăm instrucțiunile de citire și scriere în mod generalizat.

Atât cu READ cât și cu READLN se pot citi mai multe tipuri de variabile:

- de tip char;
- de tip string;
- de tip numeric.

Forma lor generală este:

READ(var nume fișier,v1[,v2,...,vn]);

READLN(var nume fișier,v1[,v2,...,vn]).

Variabilele v1...vn sunt de tipurile prezentate anterior.

Procedura READLN se execută la fel ca READ după care se face saltul la linia următoare.

Pentru a înțelege exact modul de citire (scriere) a acestor variabile, să ne imaginăm întregul fișier text (ca înșiruire de caractere ASCII). De asemenea, ne imaginăm o săgeată (se numește de altfel **pointer**) care ne indică în permanență la ce caracter am ajuns. Când deschidem un fișier pentru citire, pointer-ul indică primul caracter din prima linie a acestuia. Considerăm, de asemenea, mulțimea caracterelor ASCII împărțită în două submulțimi:

- submulțimea caracterelor cu rol de separator (spațiu, CR, LF, TAB, BACKSPACE, BELL).
- submulțimea alcătuită din restul caracterelor.

8.1.2. Citirea variabilelor de tip char

Se citește acel caracter care se găsește pe poziția curentă a pointer-ului (nu interesează dacă este vorba de caractere care au sau nu rol separator. După citire, pointer-ul avansează în fișier cu un caracter. Că este așa, o demonstrează programul următor:

```
program txt5;
var   f:text;
      a:char;

begin
   assign(f,'f1.dat');
   reset(f);
   while not eof(f) do
   begin
      read(f,a);
      write(a)
   end
end.
```

Acest program listează fișierul creat la începutul acestui capitol. Liniile apar listate exact în ordinea în care au fost introduse (se știe că f1.dat a fost creat pe mai multe linii).

8.1.3. Citirea variabilelor de tip string

Aceste variabile se citesc începând din poziția curentă a cursorului până este citit numărul de caractere necesar tipului sau până s-a ajuns la sfârșit de linie. Programul următor demonstrează acest fapt (dacă linia 1 a fișierului are 3 caractere, se vor tipări două pe un rând și unul pe al doilea rând). După citirea unei astfel de variabile, pointer-ul se află sau pe caracterul ce urmează după ultimul caracter citit sau pe CR. În situația în care pointer-ul se află pe CR și se forțează o nouă citire de tip string, se returnează șirul vid.

```
program txt6;
var   f:text;
      a:string[2];
      b:string;

begin
   assign(f,'f1.dat');
   reset(f);
   read(f,a);
   writeln(a);
   read(f,b);
   writeln(b)
end.
```

8.1.4. Citirea variabilelor de tip numeric

Se pleacă de la poziția curentă a pointer-ului. Dacă se întâlnește un caracter separator, pointer-ul sare la următorul caracter și procedeul se repetă până s-a ajuns la un caracter ce nu este separator sau până când s-a ajuns la sfârșit de linie. În al doilea caz, se solicită o nouă linie iar pointer-ul sare la începutul liniei următoare. În primul caz, se testează caracterele cuprinse între poziția curentă a pointer-ului și primul caracter separator. Dacă acestea se încadrează în diagrama de sintaxă pentru întreg (dacă se citește un tip întreg) sau real (dacă se citește un tip real), citirea se face, iar pointer-ul se așează pe primul caracter separator; dacă sintaxa nu este îndeplinită, programul se încheie cu eroare.

Exemple:

Avem următoarele variabile a:integer, f:real, e:string, d:string[2].

- Se execută read(f,a,f,e); pointer-ul este poziționat pe primul caracter din linie, iar linia este bbb12b34bbbbt (prin b am notat caracterul blanc). În urma execuției vom avea: a = 12, f = 34, e = bbbbt.
- Se execută read(f,d,e,f), prima linie din fișier este bm iar a doua 2b3 (prin b am notat blanc). Variabila d va conține bm, e va conține șirul vid, iar f va fi 2.
- Se execută read(a) iar prima linie din fișier va fi 3t (pointer-ul va fi poziționat pe primul caracter din linie). Execuția se încheie cu eroare (a nu respectă

tipul numeric).

Procedura Readln fără parametri de tip variabilă produce saltul pe linia următoare (exemplu READLN (F)), iar cu parametri, după ce a reușit tentativa de citire se sare pe linia următoare.

Pentru scriere, se folosesc procedurile WRITE și WRITELN. Forma generală a acestor proceduri WRITE este:

WRITE(var nume fișier,v1[,v2,...,vn])

unde v1, v2,...,vn reprezintă nume de variabile sau date ce se scriu în fișier. Procedura WRITELN are forma generală:

WRITELN(var nume fișier,v1[v2,...,vn]).

Parametrii au aceeași semnificație ca la procedura WRITE. Spre deosebire de WRITE, după scriere procedura WRITELN închide linia în fișier (scrie CR și LF) și trece pe linia următoare (pointer-ul este poziționat pe primul caracter al liniei următoare). Se pot scrie date de tipul:

- numeric (întreg sau real);
- char;
- string;
- boolean.

Scrierea se poate face în două feluri:

- cu format implicit;
- cu format explicit.

8.1.5. Scrierea cu format implicit

În toate aceste cazuri, se scrie începând din poziția curentă a pointer-ului.

- datele întregi se scriu cu toate cifrele semnificative, iar dacă numărul este negativ, se scrie în față semnul –;
- datele reale se scriu în forma științifică (*exemplu:* –3.35 se scrie –3.3500000000E+00, –0.7 se scrie –7.0000000000E–01) care se citește ca număr (cu semn)*10 la puterea (cu semn);
- datele de tip caracter se scriu pe o singură poziție;
- datele de tip string se scriu pe lungimea efectivă a șirului;
- pentru o dată de tip boolean se scrie șirul TRUE sau FALSE.

8.1.6. Scrierea cu format explicit

8.1.6.1. Scrierea datelor de tip întreg

Pentru fiecare variabilă întreagă (prezentă în instrucțiunea de scriere) se poate preciza numărul m de poziții pe care se face scrierea.

Exemplu: WRITE(f,a:6).

În situația în care numărul de poziții ocupate de număr este mai mare decât numărul m, parametrul este ignorat, iar dacă m este mai mare decât numărul de poziții ocupate efectiv, numărul se scrie aliniat dreapta (se va ține cont de faptul că un număr negativ va fi precedat de semn).

8.1.6.2. Scrierea datelor de tip real

Pentru aceste date se pot preciza doi parametri m și n.

Exemplu: WRITE (f,a:5:2),

unde a este o variabilă reală, m are valoarea 5 iar n are valoarea 2. Parametrul m specifică numărul de poziții pe care se scrie data reală. Parametrul n specifică numărul de poziții rezervate părții zecimale (exclusiv punctul). În cazul în care sunt prezenți ambii parametri, numărul se scrie în forma cunoscută nouă și anume cu punct (de exemplu 23.67). Dacă valoarea parametrului m este prea mică (numărul ocupă mai multe poziții), ea este ignorată (se scrie numărul cu toate cifrele lui). Dacă valoarea parametrului n este prea mică (numărul are mai multe zecimale), numărul va fi scris rotunjit cu un număr de zecimale egal cu valoarea lui n. În cazul în care numărul ocupă mai puține cifre decât valoarea lui m, acesta se scrie pe m poziții aliniat dreapta (în față se pun blancuri). Dacă parametrul n lipsește, numărul este scris în forma științifică. În acest caz, dacă parametrul m are o valoare prea mică, numărul este rotunjit, iar dacă are o valoare prea mare, el este scris aliniat dreapta.

8.1.6.3. Scrierea datelor de tip caracter

Pentru aceste date se poate folosi numai parametrul m care semnifică numărul de poziții pe care se scrie data. Dacă m este mai mare decât 1, data se scrie aliniată dreapta iar în față se pun blancuri.

8.1.6.4. Scrierea datelor de tip string

Pentru date de acest tip, se iau în considerație lungimea efectivă a șirului și valoarea parametrului m ce specifică numărul de poziții pe care se face scrierea. În cazul în care valoarea lui m este mai mică decât lungimea efectivă a șirului, aceasta se ignoră, șirul va fi scris pe lungimea sa efectivă. Dacă valoarea lui m este mai mare decât lungimea efectivă a șirului, acesta este scris pe m poziții aliniat dreapta, iar în față se pun blancuri.

8.1.6.5. Scrierea datelor de tip boolean

Avem numai două astfel de date, TRUE și FALSE. Și pentru date de acest tip se poate preciza numărul de poziții pe care să se facă scrierea. Dacă

numărul de poziții este mai mic, se ignoră, iar dacă este mai mare, acestea se scriu aliniat dreapta, în față se pun blancuri (ex. WRITE(f,TRUE:9)).

8.1.7. Alte proceduri și funcții care lucrează cu fișiere text

8.1.7.1. Procedura APPEND

Această procedură are rolul de a deschide un fișier text care a fost creat pentru extindere (scriere la sfârșit de fișier). Forma generală este APPEND(var fișier text).

În situația în care fișierul nu există, se generează o eroare de intrare ieșire. Pentru un fișier deschis cu APPEND este posibilă numai operația de scriere. În exemplul care urmează, se adaugă alte linii fișierului f1.dat.

```
program txt;
var f:text;
    a:string;

begin
  assign(f,'f1.dat');
  append(f);
  while not eof do
  begin
    readln(a);
    writeln(f,a)
  end;
  close(f)
end.
```

8.1.7.1. Funcția EOLN

Are ca parametru formal o variabilă fișier și este de tip boolean. La execuție, funcția întoarce valoarea TRUE dacă pointer-ul este la sfârșit de linie și FALSE în caz contrar.

În programul care urmează se tipărește ultimul caracter al primei linii pentru un fișier text creat anterior.

```
program txt7;
var f:text;
    a:char;
begin
assign(f,'f1.dat');
reset(f);
while not eoln(f) do   read(f,a);
writeln(a);
close(f)
end.
```

8.1.8. Fişiere text standard

8.1.8.1. Fişierele INPUT şi OUTPUT

Aceste fişiere se consideră declarate (nu mai este necesară declararea lor ca fişiere text). Ele se asignează în mod automat la tastatură (**INPUT**) şi monitor (**OUTPUT**). În situaţia în care procedurile READ, READLN, WRITE, WRITELN nu au precizată la apel variabila fişier, aceasta este considerată automat INPUT pentru procedurile de citire şi OUTPUT pentru procedurile de scriere.

Toate operaţiile de citire şi scriere efectuate în programele din capitolele anterioare au fost făcute cu aceste fişiere.

8.1.8.2. Fişierul PRN

Dacă dorim ca rezultatele programului nostru să fie scrise la imprimantă, folosim fişierul standard **PRN**. Acesta se declară în cadrul procedurii ASSIGN ca nume extern pentru fişier.

Vom lua, ca exemplu, o imprimantă care scrie pe hârtie de format A4 (coală normală de scris). Se consideră că pe o astfel de coală se pot scrie 55 de rânduri. În programul care urmează vom scrie în primele două rânduri texte, după care se sare la pagină nouă (se scriu 53 de rânduri vide cu ajutorul procedurii WRITELN).

```
program txt;
var imp:text;
    i:integer;

begin
  assign(imp,'prn');
  rewrite(imp);
  writeln(imp,'am scris un text');
  writeln(imp,'am scris alt text');
  for i:=1 to 53 do writeln(imp);
  close(imp);
end.
```

Regulile prin care se scriu diversele date la imprimantă sunt aceleaşi ca pentru oricare alt fişier text.

8.2. Validarea operaţiilor de intrare / ieşire

Din categoria operaţiilor de intrare / ieşire fac parte citirile, scrierile în şi din diverse fişiere (am văzut că monitorul şi tastatura sunt tot fişiere) dar şi operaţii cum ar fi deschiderea unui fişier pentru citire sau scriere. Nu întotdeauna aceste operaţii se desfăşoară în mod normal. Să ne imaginăm următoarea situaţie: programul nostru citeşte o variabilă numerică şi noi, din greşeală

introducem o literă. Programul se va opri și va da un mesaj de eroare. Poate veți spune că trebuie să fim atenți la introducerea datelor. Dar sunt situații în care erorile sunt inerente. De exemplu, pentru a crea un fișier ce se folosește în practică (și nu unul didactic, care are doar câteva linii) se introduc mii de date. Este aproape imposibil ca la introducerea lor să nu se greșească – caz în care ar trebui să o luăm de la capăt. Ce este de făcut pentru preîntâmpinarea acestor situații? Limbajul Turbo Pascal ne permite să prevedem în program astfel de cazuri și să acționăm în consecință.

Presupunem că programul pe care l-am conceput nu a prevăzut posibilitatea ca operațiile de intrare / ieșire să nu se desfășoare în mod corespunzător. În acest caz programul se încheie cu eroare (se afișează codul erorii). Într-un prim pas, este necesar ca programul să nu se oprească într-o astfel de situație. Pentru aceasta există o posibilitate și anume **directiva de compilare {$i}** ce poate avea două valori {$i-} și {$i+}. Directiva {$i-} determină ca toate operațiile de intrare / ieșire cuprinse între ea și directiva {$i+} sau sfârșitul programului să nu determine întreruperea acestuia.

Ne putem întreba ce realizăm dacă procedăm astfel, pentru că oricum rezultatele vor fi eronate. Există posibilitatea să determinăm dacă operația de intrare / ieșire a reușit sau nu. Funcția **IORESULT** (funcție de tip byte care se găsește prezentă în unit-ul SYSTEM) ne ajută în acest sens. În cazul unei operații reușite, la execuție funcția ia valoarea 0, în caz contrar ia o valoare diferită de 0. Prin testul valorii pe care o furnizează această funcție avem posibilitatea să reluăm operația până când ea reușește, sau să lăsăm pe cel care utilizează programul nostru să decidă ce este de făcut. În cele ce urmează, vom da mai multe exemple care au menirea să ne convingă de utilitatea de a lucra în acest mod.

Exemplul 1:

Programul v1 citește o dată numerică. În situația în care aceasta este introdusă greșit (se introduce o literă) citirea se reia.

```
program v1;
var a:integer;

begin
repeat
  write('a=');
  {$i-} readln(a) {$i+}
  until ioresult=0
end.
```

Exemplul 2:

Să presupunem că dorim crearea unui fișier text. Deschiderea lui pentru scriere se face utilizând procedura REWRITE. În situația în care pe suport se găsește alt fișier cu același nume, acesta din urmă este șters. Este bine să fim avertizați în acest sens. Prin urmare, se încearcă deschiderea fișierului care urmează a fi creat (prin RESET). Dacă el se găsește pe suport, funcția IORESULT va lua valoarea 0 iar în caz contrar va lua valoarea 1. În cazul în care aceasta ia valoarea 0, operatorul va decide dacă fișierul se va crea (deci cel vechi se distruge) sau nu.

```
program v2;
var f:text;
    a:string;
    rasp:char;

begin
  write('Dati numele fisierului ce se va crea  ');
  readln(a);
  assign(f,a);
  {$i-} reset(f) {i+};
  if ioresult<>0 then
  begin
    write (Fisierul se gaseste pe suport. Continuam ? (y/n)');
    readln(rasp)
  end;
  if rasp='y' then
  begin
    writeln ('Fisierul a fost sters');
    rewrite(f)
  end
end.
```

Exemplul 3:

Dorim să scriem un text la imprimantă. Există mai multe motive pentru care tentativa noastră poate eșua:

- imprimanta nu este deschisă;
- imprimanta nu este ON LINE;
- nu am pus hârtie.

În toate aceste cazuri, un program neprotejat se oprește. Programul care urmează scrie mesajul până când operația de scriere reușește (IORESULT ia valoarea 0).

```
program v3;
var imp:text;

begin
  assign(imp,'prn');
  rewrite(imp);
  repeat
    {$i-} writeln(imp, 'un mesaj');
  until ioresult=0
end.
```

8.2.1. Alte tipuri de validări

8.2.1.1.Validarea naturii datelor

Din acest punct de vedere datele se pot clasifica astfel:

- date numerice (numere întregi și reale);
- date alfabetice (nu conțin cifre, sunt formate din literele mari și mici ale

alfabetului);
- date alfanumerice (conțin atât literele alfabetului cât și cifre).

Să presupunem că se dorește citirea unor nume de persoane. Evident, acestea sunt date alfabetice. Cum putem testa dacă nu cumva, din greșeală, în cadrul unor nume nu s-au introdus și cifre? Exemplele de acest tip ar putea continua.

8.2.1.2. Testarea naturii numerice a datelor

O modalitate de test numeric a fost deja prezentată (prin introducerea unei litere la citirea unei variabile numerice funcția IORESULT ia valoarea 0).

Există și o altă modalitate de a efectua acest test. Pentru aceasta se procedează astfel:
- citirea datei se face într-o variabilă de tip string (deci se pot introduce orice fel de date);
- se face o tentativă de conversie din tipul string în tipul numeric solicitat (procedura **VAL**);
- în situația în care tentativa reușește, data introdusă a fost numerică;
- în caz contrar, data introdusă nu a fost numerică deci se reia citirea.

Această modalitate de lucru prezintă și marele avantaj de a ne oferi posibilitatea să testăm numărul de cifre introduse, prin analiza lungimii șirului de tip string (funcția **LENGTH**).

În programul care urmează se citește o dată care trebuie să fie numerică și să fie scrisă pe trei cifre (care pot fi și 0).

```
program v4;
var a:string;
    c_er,b:integer;

begin
  repeat
    write('b=');
    readln(a);
    val(a,b,c_er)
  until (c_er=0)and(length(a)=3)
end.
```

8.2.1.3. Testarea naturii alfabetice a datelor

Pentru acest test se procedează astfel:
- literele mari și mici ale alfabetului alcătuiesc o mulțime și se declară constante;
- citirea se face într-o variabilă de tip string;
- se testează toate caracterele șirului citit;
- dacă cel puțin un caracter nu aparține mulțimii considerate, citirea se reia.

În programul care urmează se exemplifică acest test.

```
program v5;
const alfabet=['a'..'z','A'..'Z'];
var a:string;
    i,c_er:integer;

begin
  repeat
    c_er:=0;
    write('dati data ');
    readln(a);
    for i:=1 to length(a) do
      if not (a[i] in alfabet then c_er:=1
  until c_er=0
end.
```

Datele alfanumerice nu necesită validare.

Toate tipurile de validare prezentate aici sunt valabile indiferent de tipul de fișier ales. Din acest motiv, în cele ce urmează le folosim fără explicații suplimentare.

8.3. Fișiere cu tip

Fișierele cu tip sunt construite din articole cu lungime fixă, de același tip, motiv pentru care se mai numesc și fișiere cu prelucrare la nivel de articol. Tipul fișier cu tip se declară conform figurii de mai jos:

Figura 8.3.1.

Exemple de declarare:

- Se declară două fișiere ale căror articole sunt constituite dintr-un singur octet:

```
type fisoct=file of byte
var f1,f2:fisoct;
```

Aceleași două fișiere se pot declara printr-un tip anonim:

```
var f1,f2 : file of byte
```

- Se declară un fișier ale cărui articole sunt numere întregi:

```
type fisint=file of integer;
.............................
var a:fisint;
```

- Se declară un fișier cu articole de tip record:

```
type  inreg=record
      nume:string[30];
      vârsta:byte
      end;
      fișier=file of inreg;
var f:fișier;
```

Ca și pentru fișierele text, procedura ASSIGN face legătura între numele fișierului din program și numele său de pe suportul extern. Forma generală este:

ASSIGN(nume fișier în program, 'nume extern').

Numele extern este de tipul string.

Exemplu: assign (f,'fișier'); unde f reprezintă numele variabilei fișier în program iar 'fișier' reprezintă numele sub care acesta se găsește pe suport.

8.3.1. Crearea fișierelor cu tip

Pentru a putea realiza o astfel de operație, trebuie cunoscute câteva proceduri și funcții.

Procedura **REWRITE** are forma generală:

REWRITE(var fișier)

și are rolul de a deschide un fișier spre a fi creat. În situația în care mai există un fișier cu același nume, acesta se distruge. Din acest motiv, este bine să se facă testul de existență (vezi paragraful anterior).

Procedura **WRITE** are forma generală:

WRITE(var fișier,v1...vn)

Parametrii v1...vn reprezintă variabile ce vor fi scrise în fișier. Principiul de execuție este următorul:

- se scrie prima înregistrare preluată din cadrul variabilei v1;
- se scrie a doua înregistrare preluată din cadrul variabilei v2;

.
.
.

- se scrie înregistrarea n din cadrul variabilei vn;
- pointer-ul se poziționează pe următoarea înregistrare.

Funcția booleană **EOF** are forma generală:

EOF (var fișier)

și returnează TRUE dacă pointer-ul se găsește la sfârșit de fișier, în caz contrar returnează FALSE.

Procedura **CLOSE** are forma generală:

CLOSE(var fișier)

și are rolul de a închide fișierul.

În programul care urmează, după fiecare înregistrare citită de la tastatură și scrisă în fișier, utilizatorul este întrebat dacă să se continue sau nu. După ce au fost scrise toate înregistrările, fișierul se închide.

```
program ft1;
type inr= record
            nume:string[10];
            vârsta:byte
          end;
fisier=file of inr;
var inreg:inr;
    f1:fisier;
    c:char;

begin
  assign(f1,'fdat');
  rewrite(f1);
  repeat
    write('nume ');
    readln(inreg.nume);
    write('vârsta ');
    readln(inreg.vârsta);
    write(f1,inreg);
    write('continuati ? (y/n) ');
    readln (c);
  until c='n';
  close(f1)
end.
```

Evident, nu este obligatoriu ca un fișier să se creeze de la tastatură, el poate fi creat utilizând informații preluate din alte fișiere, însă pentru aceasta trebuie să știm cum se citește un fișier.

Pentru a fi citit, un fișier se deschide cu ajutorul procedurii **RESET**, care are forma generală:

RESET(var fișier).

Procedura **READ** are rolul de a citi una sau mai multe înregistrări ale unui fișier. Forma generală a acestei proceduri este :

READ(var fișier,v1,v2,...,vn).

Principiul de execuție este următorul:

- se citește înregistrarea pe care se află pointer-ul în variabila v1;
- pointer-ul trece pe următoarea înregistrare care este citită în variabila v2;
- procedeul continuă până când este citită variabila vn și pointer-ul se plasează pe următoarea înregistrare necitită sau pe marcajul de sfârșit de fișier.

Programul care urmează citește fișierul creat anterior și îl tipărește pe monitor.

```
program ft2;
type inr= record
            nume:string[10];
            vârsta:byte
          end;
fișier=file of inr;
var inreg:inr;
    f1:fisier;

begin
  assign(f1,'fdat');
  reset(f1);
  while not eof(f1) do
  begin
    read (f1,inreg);
    writeln(inreg.nume, ' ',inreg.vârsta)
  end;
  close(f1)
end.
```

Aplicație:

Concatenare de fișiere

Dacă se dau două fișiere, prin operația de **concatenare** a lor se înțelege crearea cu ajutorul lor a unui al treilea fișier care conține înregistrările celor două în odinea următoare: înregistrările primului fișier urmate de înregistrările celui de-al doilea fișier.

Programul care urmează concatenează două fișiere. În esență, pentru a realiza această operație, se procedează astfel:

- se deschid toate fișierele (două pentru citire, unul pentru scriere);
- toate înregistrările primului fișier se copiază în cadrul celui de-al treilea;
- toate înregistrările celui de-al doilea fișier se copiază, în continuare, în cadrul celui de-al treilea;
- în final, se închid cele trei fișiere.

```
program ft3;
type inr= record
            nume:string[10];
            vârsta:byte
          end;
fisier=file of inr;
var inreg:inr;
    f1,f2,f3:fisier;

begin
  assign(f1,'fdat');
  assign(f2,'fdat1');
  assign(f3,'fdat2');
  reset(f1);
  reset(f2);
  rewrite(f3);
  while not eof(f1) do
  begin
```

```
    read (f1,inreg);
    write(f3,inreg);
  end;
  while not eof(f2) do
  begin
    read(f2,inreg);
    write(f3,inreg);
  end;
  close(f1);
  close(f2);
  close(f3);
end.
```

8.3.2. Procedura SEEK şi funcţia FILESIZE

Înregistrările unui fişier cu tip au aceeaşi lungime. Fiecărei înregistrări i se poate asocia un număr de ordine cuprins între 0 (pentru prima înregistrare) şi n–1 (pentru înregistrarea n). Datorită lungimii fixe a înregistrărilor, acestea pot fi accesate direct (se poate poziţiona pointer-ul pe înregistrarea cu un anumit număr de ordine). Este firesc să fie posibil acest lucru, întrucât dacă se doreşte să poziţionăm pointer-ul pe înregistrarea i, adresa de început a acesteia (a primului ei octet) se obţine înmulţind lungimea unei înregistrări cu i (*Atenţie!* înregistrările sunt numerotate între 0 şi n–1).

Procedura **SEEK** are forma generală:

SEEK(var fişier,număr înregistrare)

şi are rolul de a poziţiona pointer-ul pe înregistrarea cu numărul de ordine 'număr înregistrare'.

Funcţia **FILESIZE** având forma generală:

FILESIZE (var fişier)

este tip longint şi furnizează numărul de componente pe care le are fişierul. Dacă din rezultatul furnizat de această funcţie se scade 1, obţinem numărul de ordine al ultimei componente din fişier.

În programul care urmează, se listează un fişier utilizând funcţia FILESIZE

```
program ft4;
type inr= record
            nume:string[10];
            vârsta:byte
          end;
fisier=file of inr;
var inreg:inr;
    f1:fisier;
    i:integer;

begin
  assign(f1,'fdat2');
  reset(f1);
  for i:=0 to filesize(f1)-1 do
```

```
  begin
    read (f1,inreg);
    writeln(inreg.nume, ' ',inreg.vârsta)
  end;
  close(f1)
end.
```

8.3.3. Adăugarea de articole

Fiind dat un fișier, se cere să-i adăugăm un număr de articole. În programul următor, adăugarea articolelor se face la sfârșitul fișierului (deci după toate articolele conținute deja în fișier).

Pentru a realiza acest lucru se poziționează pointer-ul pe marcajul sfârșitului de fișier (seek(f1,filesize(f1)), iar înregistrările care se adaugă sunt citite de la tastatură și scrise în continuare. Se observă că deschiderea fișierului a fost făcută cu RESET (o deschidere cu RESET permite și scrierea în fișier).

```
program ft5;
type inr= record
            nume:string[10];
            vârsta:byte
          end;
fisier=file of inr;
var inreg:inr;
    f1:fisier;
    dfghjkl;c:char;

begin
  assign(f1,'fdat2');
  reset(f1);
  seek(f1,filesize(f1));
  repeat
    write('nume ');
    readln(inreg.nume);
    write('vârsta ');
    readln(inreg.vârsta);
    write(f1,inreg);
    write('continuati ? (y/n) ');
    readln (c);
  until c='n';
  close(f1)
end.
```

De multe ori, adăugarea noilor înregistrări trebuie făcută în interiorul fișierului (după o anumită înregistrare). Limbajul Turbo Pascal nu permite ca această operație să se facă în mod direct. Din acest motiv, pentru a realiza ceea ce dorim se procedează în felul următor:

- se deschide un al doilea fișier;
- se copiază în acesta toate înregistrările care se găsesc plasate înaintea celor care se adaugă;
- se scriu, în continuare, noile înregistrări;

- se copiază restul înregistrărilor (din vechiul fișier);
- vechiul fișier este șters, iar noul fișier capătă numele vechiului fișier.

Programul care urmează citește de la tastatură înregistrările ce se adaugă. Pentru ștergere, se folosește procedura **ERASE** care are forma generală:

ERASE(var fișier)

Iată deci că un fișier poate fi șters și din interiorul unui program. Redenumirea unui fișier este făcută de procedura RENAME având forma generală:

RENAME (var fișier, 'nume')

unde 'nume' este un parametru de tip string și reprezintă noul nume al fișierului.

```
program ft6;
type inr= record
            nume:string[10];
            vârsta:byte
          end;
fisier=file of inr;
var inreg:inr;
    f1,f2:fisier;
    c:char;
    i:integer;

begin
  assign(f1,'fdat2');
  reset(f1);
  assign(f2,'fis1');
  rewrite(f2);
  for i:=1 to 3 do
  begin
    read(f1,inreg);
    write(f2,inreg);
  end;
  repeat
    write('nume ');
    readln(inreg.nume);
    write('vârsta ');
    readln(inreg.vârsta);
    write(f2,inreg);
    write('continuati ? (y/n) ');
    readln (c);
  until c='n';
  while not eof(f1) do
  begin
    read(f1,inreg);
    write(f2,inreg)
  end;
  close(f1);
  close(f2);
  erase (f1);
  rename(f2,'fdat2')
end.
```

De multe ori, este necesar să modificăm un câmp (sau mai multe) al unei înregistrări (sau ale mai multor înregistrări). Programul ft7 face modificare în câmpul vârstă al înregistrării cu numele de ordine 4. Pentru aceasta, se procedează în felul următor:

- după deschiderea fișierului (cu RESET), poziționăm pointer-ul pe înregistrarea care urmează a fi modificată (cu SEEK);
- se citește înregistrarea;
- se face modificarea în câmpul corespunzător;
- întrucât, la citire, pointer-ul s-a poziționat pe înregistrarea următoare, acesta se repoziționează pe înregistrarea care a fost citită;
- această înregistrare este suprascrisă (câmpul a fost modificat) cu procedura WRITE.

```
program ft7;
type inr= record
            nume:string[10];
            vârsta:byte
          end;
fisier=file of inr;
var inreg:inr;
    f1:fisier;

begin
  assign(f1,'fdat2');
  reset(f1);
  seek(f1,4);
  read(f1,inreg);
  inreg.vârsta:=70;
  seek(f1,4);
  write(f1,inreg);
  close(f1)
end.
```

Cum putem șterge una sau mai multe înregistrări ale fișierului? În practică, pentru aceasta se poate proceda în două feluri:

- înregistrările se șterg fizic;
- înregistrările se șterg logic.

Ștergerea fizică nu este posibilă decât prin a crea un nou fișier care să nu conțină înregistrările care se șterg. Vechiul fișier este șters iar noul fișier capătă numele fișierului inițial (cel din care s-au șters înregistrări). Programul care urmează șterge înregistrarea a patra din fișier.

Din păcate, această modalitate de ștergere cere mult timp (pentru a șterge o înregistrare am creat un alt fișier).

```
program ft8;
type inr= record
            nume:string[10];
            vârsta:byte
          end;
fisier=file of inr;
var inreg:inr;
    f1,f2:fisier;
```

```
    i:integer;

begin
  assign(f1,'fdat2');
  assign(f2,'fman');
  reset(f1);
  rewrite(f2);
  for i:=1 to 3 do
  begin
    read(f1,inreg);
    write(f2,inreg)
  end;
  seek(f1,filepos(f1)+1);
  while not eof(f1) do
  begin
    read(f1,inreg);
    write(f2,inreg)
  end;
  close(f1);
  close(f2);
  erase(f1);
  rename(f2,'fdat2')
end.
end.
```

Ştergerea logică are la bază următorul mecanism: fiecare înregistrare a fişierului în care urmează să se facă ştergeri are un câmp în plus şi anume un indicator de ştergere (is). Convenim ca acest câmp să conţină valoarea 1 în situaţia în care înregistrarea nu este ştearsă şi să conţină valoarea 0 în caz contrar. În acest fel, a şterge o înregistrare înseamnă să modificăm câmpul is (va lua valoarea 0). Această modalitate de ştergere este mult mai rapidă, dar prezintă dezavantajul că înregistrările şterse rămân pe suport (se ocupă spaţiu).

Următoarele trei programe creează, listează şi efectuează ştergeri pentru un fişier.

Observaţie: orice program care lucrează cu un astfel de fişier trebuie să testeze, pentru fiecare înregistrare, valoarea indicatorului de ştergere (nu vor fi supuse prelucrării înregistrări şterse).

```
program ft9;
type inr= record
            is:byte;
            nume:string[10];
            vârsta:byte;
          end;
fisier=file of inr;
var inreg:inr;
    f1:fisier;
    c:char;

begin
  assign(f1,'fdat2');
  rewrite(f1);
  repeat
```

```
    write('nume ');
    readln(inreg.nume);
    write('vârsta ');
    readln(inreg.vârsta);
    inreg.is:=1;
    write(f1,inreg);
    write('continuati ? (y/n) ');
    readln(c);
  until c='n';
  close(f1)
end.
```

```
program ft10;
type inr= record
            is:byte;
            nume:string[10];
            vârsta:byte
          end;
fisier=file of inr;
var inreg:inr;
    f1:fisier;
    c:char;

begin
  assign(f1,'fdat2');
  reset(f1);
  while not eof(f1) do
  begin
    read(f1,inreg);
    if inreg.is=1 then writeln(inreg.nume, ' ',inreg.vârsta)
  end;
  close(f1)
end.
```

```
program ft11;
type inr= record
            is:byte;
            nume:string[10];
            vârsta:byte;
          end;
fisier=file of inr;
var inreg:inr;
    f1:fisier;

begin
  assign(f1,'fdat2');
  reset(f1);
  seek(f1,5);
  read(f1,inreg);
  seek(f1,filepos(f1)-1);
  inreg.is:=0;
  write(f1,inreg);
  close(f1)
end.
```

Se consideră un fişier. Se cere ca acesta să fie sortat crescător (sau descrescător) după valorile unui anumit câmp. În programul care urmează, fişierul se sortează după valoarea câmpului nume (sortarea alfabetică).

Pentru a realiza sortarea, se folosește „metoda bulelor" care a fost prezentată în cadrul acestui manual. În esență, se parcurge fișierul de mai multe ori și se inversează înregistrările alăturate care nu sunt în ordinea cerută. Dacă la o parcurgere s-a efectuat cel puțin o inversare, fișierul se parcurge din nou.

```
program ft12;
type inr= record
            nume:string[10];
            vârsta:byte
          end;
fisier=file of inr;
var f1:fisier;
    i:integer;
    inv:boolean;
    inr1,inr2:inr;

begin
  assign(f1,'fdat2');
  reset(f1);
  repeat
    inv:=false;
    for i:=0 to filesize(f1)-2 do
    begin
      seek(f1,i);
      read(f1,inr1,inr2);
      if inr1.nume>inr2.nume then
      begin
        seek(f1,i);
        write(f1,inr2,inr1);
        inv:=true
      end
    end
  until not inv;
  close(f1)
end.
```

8.4. Un exemplu de lucru cu un fișier

Se cere să se scrie un program care să permită prelucrarea pe calculator a rezultatelor unui examen de admitere la liceu. Acest program trebuie să creeze, să actualizeze și să listeze un fișier ce conține rezultatele obținute de fiecare candidat. Fiecare înregistrare trebuie să conțină :

- numele elevului;
- nota obținută la proba 1;
- nota obținută la proba 2;
- media generală.

Această aplicație urmărește să demonstreze complexitatea programelor care se folosesc în practică. Desigur, există limbaje specializate pentru scrierea aplicațiilor de acest gen (DBASE, FOXPRO), însă un elev la o clasă cu profil

informatică trebuie să aibă cunoștințe care să-i permită o înțelegere profundă a ceea ce se întâmplă de fapt și nu doar cunoștințe de suprafață.

Programul realizează următoarele:
- crearea fișierului;
- adăugarea unor înregistrări;
- ștergere de înregistrări;
- modificarea înregistrărilor;
- sortarea fișierului în ordinea descrescătoare a mediilor;
- listarea lui pe monitor, în vederea verificării informațiilor;
- listarea la imprimantă a candidaților admiși.

În afara acestora, programul trebuie să realizeze și alte operații cum ar fi:
- listarea la imprimantă a candidaților respinși;
- listarea pe monitor atât a candidaților admiși, cât și a celor respinși.

Aceste funcții nu sunt realizate, le propunem ca exercițiu.
La sfârșitul acestui paragraf prezentăm schema de principiu a aplicației.

Apelul procedurilor care realizează funcțiile specificate este realizat de procedura **APLIC**, prezentă în cadrul unității de program **FIS**. Aceasta creează un meniu prin utilizarea unității MENU prezentată în paragraful precedent.
În afara procedurilor care realizează funcțiile specificate și care vor fi explicate ulterior, mai este apelată o procedură numită DIALOG1.

Procedura **DIALOG1** cere numele fișierului de candidați cu care se va lucra în continuare. Procedura testează dacă acest fișier se găsește sau nu pe suportul magnetic. Rezultatul este afișat pe ecran. Pentru aceasta se procedează astfel:
- se încearcă deschiderea pentru citire a fișierului cu nume specificat (opțiunile de compilare {$i-} și {$i+} ce încadrează procedura RESET au rolul de a inhiba o eventuală oprire a programului ca urmare a faptului că se încearcă deschiderea spre citire a unui fișier inexistent iar funcția IORESULT returnează valoarea 0 în cazul unei operații de intrare / ieșire reușite, altfel returnează o valoare diferită de 0);
- în funcție de reușita sau nu a operației, se afișează mesajul corespunzător;
- în cazul în care fișierul a fost deschis spre citire (deci se găsește pe suport) este închis;
- numele fișierului este reținut în variabila nume_f, variabilă la care au acces toate procedurile din unitatea de program.

Procedura **CREARE** are rolul de a crea fișierul cu candidați. Numele său este preluat din variabila nume_f. Pentru aceasta se procedează astfel:
- pentru citirea informațiilor de pe ecran se folosește procedura CITIRE_TAST;
- media generală care este conținută în înregistrare se calculează înainte de a scrie înregistrarea în fișier;
- după fiecare înregistrare, utilizatorul este întrebat dacă dorește sau nu să continue.

Procedura **CITIRE_TAST** apelează repetitiv procedura INTERFATA (spre a citi datele de pe ecran) până când utilizatorul consideră că datele tastate sunt corecte.

Procedura **INTERFATA** realizează următoarele:
- șterge ecranul;

- afișează numărul înregistrării curente (care urmează să fie citită);
- afișează în locuri fixe numele informațiilor ce urmează a fi citite;
- citește în locuri fixe informațiile, procedând în felul următor:
  - citirea propriu-zisă se face în variabile de tip string;
  - în cazul notelor, care sunt numere reale, citirea se face în mod repetitiv până când sunt îndeplinite simultan condițiile:
    - conținutul tipului string în care au fost citite se poate converti fără eroare în variabile reale (pentru aceasta se utilizează procedura VAL cu cei trei parametri: numele variabilei de tip string, numele variabilei reale în care urmează să se facă conversia și numele variabilei care reține dacă operația a reușit, caz în care ia valoarea 0, altfel ia o valoare diferită de 0);
    - notele tastate sunt mai mari ca 0 și mai mici sau egale cu 10;
    - șirul citit nu este vid (datorită apăsării din greșeală a tastei RETURN).

Procedura **ADAUGARE** are rolul de a adăuga la sfârșitul fișierului una sau mai multe înregistrări. Se procedează în felul următor:
- se deschide fișierul pentru citire;
- se află numărul de înregistrări ale fișierului (pentru aceasta se utilizează funcția FILESIZE);
- cu ajutorul funcției SEEK se face poziționarea după ultima înregistrare (pe marcajul sfârșitului de fișier);
- cu ajutorul procedurii CITIRE_TAST prezentate anterior se citesc informațiile de la tastatură;
- după calculul mediei generale, înregistrarea este scrisă;
- procedeul continuă până când utilizatorul, care este întrebat de fiecare dată dacă dorește sau nu acest lucru, răspunde negativ.

Procedura **STERGERE** are rolul de a șterge o înregistrare. În acest scop se procedează astfel:
- se cere numele candidatului ce urmează a fi șters;
- întrucât în fișiere secvențiale nu există o instrucțiune capabilă de a șterge o înregistrare, fișierul de unde urmează a fi ștearsă înregistrarea este citit iar fiecare înregistrare diferită de cea care urmează a fi ștearsă este trecută într-un fișier de manevră; în final, fișierul inițial este șters iar fișierul de manevră capătă numele fișierului inițial;
- în plus, dacă este găsită înregistrarea ce urmează a fi ștearsă, se semnalează acest lucru într-o variabilă booleană iar la sfârșit utilizatorul știe dacă acțiunea de ștergere a fost încununată de succes sau nu.

Menționăm că funcția POS are rolul de a căuta un subșir într-un șir și în cazul în care acesta este găsit ia o valoare diferită de 0, în caz contrar ia valoarea 0. Această funcție este folosită în procedură pentru a căuta numele candidatului în cadrul variabilei ce reține numele din înregistrarea citită (în felul acesta se evită cerința suplimentară ca atât numele prezent în cadrul înregistrării cât și numele tastat în procedură să ocupe aceleași poziții în cadrul variabilelor ce le corespund).

Procedura **MODIFICARE** are rolul de a modifica o înregistrare din cadrul fișierului. Se procedează astfel:
- se cere numele candidatului a cărui înregistrare urmează a fi modificată;
- se parcurge fișierul până când este găsită înregistrarea corespunzătoare sau

până când a fost citit tot fișierul;
- dacă înregistrarea a fost găsită, poziționăm pointer-ul asupra ei (utilizând funcția FILEPOS, care furnizează numărul de înregistrări deja citite și procedura SEEK pentru poziționare);
- se citesc noile informații corespunzătoare ei prin intermediul procedurii CITIRE_TAST, informații care sunt trecute în locul vechii înregistrări;
- se semnalează utilizatorului dacă înregistrarea cu numele citit în procedură nu a fost găsită.

Procedura **SORTARE** are rolul de a sorta fișierul de candidați în ordinea descrescătoare a mediilor. Pentru aceasta se folosește o metodă cunoscută sub numele de metoda bulelor. Algoritmul este următorul:
- se compară media generală a candidatului care se găsește în prima înregistrare cu media generală a candidatului ce se găsește în a doua înregistrare din fișier iar în cazul în care prima medie este mai mică decât a doua cele două înregistrări se inversează;
- se compară în mod asemănător și se inversează dacă este cazul înregistrarile a doua cu a treia;
- se compară și se inversează dacă este cazul penultima și ultima înregistrare din fișier;
- dacă în timpul parcurgerii fișierului s-a efectuat cel puțin o interschimbare de înregistrări, se reia parcurgerea fișierului, în caz contrar algoritmul se încheie iar înregistrările sale sunt sortate.

Procedura **LISTARE_P** listează pe monitor fișierul de candidați. În comentariul acestei proceduri vom preciza următoarele:
- utilizează procedura **CAPTAB** ce are rolul de a tipări capul de tabel (întrucât procedura CAPTÀB are posibilitatea de a tipări capul de tabel atât pe monitor cât și la imprimantă, aceasta este apelată cu un parametru ce ia valoarea 0 pentru monitor și valoarea 1 pentru imprimantă);
- pentru a evita consecințele neplăcute ale fenomenului de defilare pe monitor, se cere numărul de rânduri ce vor fi prezente simultan pe ecran (număr ce include și cele 4 rânduri ocupate de capul de tabel și titlul raportului);
- după fiecare ecran listat utilizatorul este întrebat dacă dorește să continue sau nu listarea.

Listarea la imprimantă a candidaților admiși se face prin apelarea a trei proceduri:
- DIALOG
- ADMRESP
- LISTARE_A.

Procedura **DIALOG** îi cere utilizatorului numărul de locuri puse la dispoziția candidaților. Dialogul cu utilizatorul se face luând măsurile de protecție următoare:
- de a nu introduce un număr greșit (pentru siguranță, utilizatorul este întrebat dacă ce a introdus este corect sau nu);
- de a inhiba o eventuală întrerupere de program ca urmare a introducerii unei litere în locul unei cifre.

Procedura **ADMRESP** calculează numărul de candidați admiși și numărul de candidați respinși. Pentru ca un candidat să fie admis este necesar să îndepli-

nească următorele condiții:
- să aibă o medie generală mai mare sau egală cu 5;
- în ordinea descrescătoare a mediilor să aibă o medie care se încadrează în numărul de locuri, sau egală cu cea a candidatului uitim admis;
- numărul de candidați respinși se calculează ca diferența dintre numărul candidaților (numărul de înregistrări din fișier) și cei admiși.

Procedura **LISTARE_A** are rolul de a lista la imprimantă candidații admiși. Ea cere utilizatorului să furnizeze numărul de rânduri ce se vor tipări pe o pagină:
- se tipărește capul de tabel utilizând procedura CAPTAB cu parametrul 1 (pentru a lista capul de tabel și titlul la imprimantă);
- informațiile corespunzătoare fiecărei înregistrări sunt convertite în variabile de tip string (cu procedura STR ce are doi parametri: numele variabilei sursă și numele variabilei destinație), iar acestea sunt inserate în variabila de tip string rând (cu ajutorul procedurii INSERT cu trei parametri și anume variabila destinație, variabila sursă și lungimea pe care să se facă transferul);
- variabila rând este tipărită;
- după fiecare pagină tipărită se așteaptă introducerea noii pagini după care procesul se reia până la tipărirea tuturor candidaților.

Procedura **CAPTAB** scrie titlul și capul de tabel pe monitor sau la imprimantă, după parametrul cu care a fost apelată (0 sau 1). Pentru scrierea titlului se apelează procedura **TITLU** cu același parametru (0 sau 1) pentru monitor sau imprimantă.

Programul utilizator care lansează această aplicație este următorul:

```
program examen;
uses fis;

begin
  aplic
end.
```

Iată și unitatea de program:

```
unit fis;

interface
uses crt,menu,util;
procedure aplic;

implementation
type inr= record
            nume:string[20];
            nota1,nota2,media:real
          end;
var n1,n2:real;
    nume,nume_f:string[20];
    tit:string[40];
    buf:string[5];
    c_er,i,nr_loc,nr_adm,nr_resp:integer;
    inreg:inr;
    f,h:file of inr;
    imprim:text;
```

```
procedure interfata;
begin
  window(1,1,80,25);
  textbackground(negru);
  textcolor(alb);
  clrscr;
  cursor(true);
  gotoxy(25,1);
  write(' înregistrarea ',i);
  gotoxy(3,3);
  write('nume candidat');
  gotoxy(3,5);
  write('nota proba 1');
  gotoxy(3,7);
  write('nota proba 2');
  repeat
    gotoxy(20,3);
    clreol;
    readln(nume)
  until nume<>'';
  repeat
    gotoxy(20,5);
    clreol;
    readln(buf);
    val(buf,n1,c_er)
  until (c_er=0) and (buf<>'') and (n1>=0) and (n1<=10);
  repeat
    gotoxy(20,7);
    clreol;
    readln(buf);
    val(buf,n2,c_er)
  until (c_er=0) and (buf<>'') and (n2>=0) and (n2<=10);
end;

procedure citire_tast;
var c:char;
begin
  repeat
    interfata;
    gotoxy(3,20);
    write('corect (y/n)');
    gotoxy(17,20);
    c:=readkey;
  until c<>'n'
end;

procedure creare;
var c: char;
begin
  rewrite(f);
  i:=0;
  repeat
    i:=i+1;
    citire_tast;
    inreg.nume:=nume;
    inreg.nota1:=n1;
```

```
      inreg.nota2:=n2;
      inreg.media:=(n1+n2)/2;
      write(f,inreg);
      gotoxy(3,21);
      write('continuati (y/n) ');
      gotoxy(20,21);
      c:=readkey;
    until c='n';
    close(f)
  end;

  procedure dialog1;
  var c:char;
  begin
    textbackground(negru);
    textcolor(alb);
    clrscr;
    gotoxy(10,4);
    write('numele fisierului');
    repeat
      gotoxy(30,4);clreol;
      read(nume_f);
      gotoxy(10,20);
      write('corect (y/n)');
      gotoxy(25,20);
      readln;
      c:=readkey;
      delline
    until c='y';
    assign(f,nume_f);
    {$i-} reset(f); {$i+}
    gotoxy(10,20);
    if ioresult=0 then
    begin
      close(f);
      gotoxy(10,20);
      write('fisierul exista ')
    end
    else write('fisierul trebuie creat ');
    delay(2000);
  end;

  procedure adaugare;
  var c:char;
  begin
    assign(f,nume_f);
    reset(f);
    i:=filesize(f);
    seek(f,i);
    repeat
      i:=i+1;
      citire_tast;
      inreg.nume:=nume;
      inreg.nota1:=n1;
      inreg.nota2:=n2;
      inreg.media:=(n1+n2)/2;
```

```
    write(f,inreg);
    gotoxy(3,21);
    write('continuati (y/n) ');
    gotoxy(20,21);
    c:=readkey;
  until c='n';
  close(f);
end;

procedure stergere;
var c:inr;
    num:string[80];
    gasit:boolean;
    a:string[1];
begin
  window(1,1,80,25);
  textbackground(negru);
  textcolor(alb);
  clrscr;
  write('numele candidatului pe care îl stergeti: ');
  readln(num);
  assign(f,nume_f);
  reset(f);
  assign(h,'fman.dat');
  rewrite(h);
  gasit:=false;
  while not eof(f) do
  begin
    read(f,c);
    if pos(num,c.nume)<>0  then gasit:=true
                           else write(h,c);
  end;
  if gasit   then writeln ('succes-înregistrare stearsa')
             else writeln ('înregistrare negasita ');
  close(f);
  close(h);
  erase(f);
  rename(h,nume_f);
  gotoxy(1,20);
  writeln('apasati`o tasta...');
  a:=readkey
end;

procedure modificare;
var c:inr;
    num:string[80];
    gasit:boolean;
    a:string[1];
begin
  window(1,1,89,25);
  textbackground(negru);
  textcolor(alb);
  clrscr;
  cursor(true);
  assign(f,nume_f);
  reset(f);
```

```
    write ('numele candidatului ce urmeaza a fi modificat ? ');
    readln(num);
    gasit:=false;
    while (not eof(f)) and (not gasit) do
    begin
      read(f,c);
      if num=c.nume then gasit:=true;
    end;
    i:=filepos(f);
    if găsit then
    begin
      citire_tast;
      c.nume:=nume;
      c.nota1:=n1;
      c.nota2:=n2;
      c.media:=(n1+n2)/2;
      seek(f,i-1);
      write(f,c);
    end
    else writeln('nu exista aceasta înregistrare');
    writeln('apasati o tasta');
    a:=readkey;
    close(f)
  end;

  procedure sortare;
  var c,d:inr;
      n,j:integer;
      inversari:boolean;
  begin
    assign(f,nume_f);
    reset(f);
    n:=filesize(f);
    repeat
      inversari:=false;
      for j:=0 to n-2 do
      begin
        seek(f,j);
        read(f,c);
        read(f,d);
        if d.media>c.media then
        begin
          seek(f,j);
          write(f,d);
          write(f,c);
          inversari:=true
        end
      end
    until not inversari;
    close(f)
  end;

  procedure titlu(t:integer);
  var c:char;
  begin
    if t=0 then
```

```
  begin
    clrscr;
    gotoxy(14,1);
    write(tit)
  end
  else
  begin
    writeln('deschideti imprimanta');
    writeln('apasati o tasta ');
    c:=readkey;
    repeat
      {$i-}
      writeln(imprim,tit);
      {$i+}
    until ioresult=0
  end;
end;

procedure captab(t:integer);
var a,b:string[60];
begin
  a:= '*************************************************';
  b:= '*nc*      nume elev      * nota1 * nota2 * media *';
  titlu(t);
  if t=0 then
  begin
    gotoxy(1,2);write(a);
    gotoxy(1,3);write(b);
    gotoxy(1,4);write(a);
  end
  else
  begin
    a:='     '+a;
    b:='     '+b;
    writeln(imprim,a);
    writeln(imprim,b);
    writeln(imprim,a);
  end
end;

procedure listare_p;
var nr_rând,j:integer;
    c:string[1];
begin
  tit:='lista de verificare';
  repeat
    window(1,1,80,25);
    textbackground(negru);
    textcolor(alb);
    clrscr;
    cursor(true);
    write('câte rânduri sa se tipareasca pe ecran ? ');
    readln(nr_rând)
  until (nr_rând>=4) and (nr_rând<=20);
  assign(f,nume_f);
  reset(f);
```

```
    c:='y';
    i:=1;
    while (not eof(f)) and (c<>'n') do
    begin
      j:=5;
      captab(0);
      while (j<=nr_rând) and (not eof(f)) and (c<>'n') do
      begin
        read(f,inreg);
        gotoxy(2,j); write(i);
        gotoxy(5,j); write(inreg.nume);
        gotoxy(26,j); write(inreg.nota1:2:2);
        gotoxy(34,j); write(inreg.nota2:2:2);
        gotoxy(42,j); write(inreg.media:2:2);
        j:=j+1;
        i:=i+1
      end;
      gotoxy(1,21);
      write('continuati? (y/n) ');
      gotoxy(20,21);
      readln(c);
    end;
    close(f)
end;

procedure listare_a;
var c:char;
    n,j,k:integer;
    rând :string[60];
    man:string[3];
    man1:string[5];

begin
  if nr_adm<>0 then
  begin
    clrscr;
    assign(f,nume_f);
    reset(f);
    repeat
      write('câte rânduri pe pagina (5..55) ');
      {$i-} readln(n) {$i+}
    until (ioresult=0) and (n>=5) and(n<=55);
    j:=1;
    assign(imprim,'prn');
    rewrite(imprim);
    repeat
      k:=5;
      tit:=' candidaţi admişi';
      captab(1);
      repeat
        read(f,inreg);
        rând:='
        man:='   ';
        man1:='     ';
        str(j,man);
        insert(man,rând,2);
```

```
        insert(inreg.nume,rând,5);
        str(inreg.nota1:2:2,man1);
        insert(man1,rând,26);
        str(inreg.nota2:2:2,man1);
        insert(man1,rând,34);
        str(inreg.media:2:2,man1);
        insert(man1,rând,42);
        rând:='    '+rând;
        writeln(imprim,rând);
        j:=j+1;
        k:=k+1
      until (k=n) or ( j=nr_adm+1)
    until j=nr_adm+1;
    close(f);
    close(imprim)
  end
end;

procedure dialog;
var c:char;
begin
  window(1,1,80,25);
  textbackground(negru);
  textcolor(alb);
  cursor(true);
  clrscr;
  repeat
    repeat
      write('câte locuri sunt ? ');
      {$i-} readln(nr_loc);{$i+}
    until  ioresult=0;
    write ('corect (y/n) ');
    readln(c);
  until c='y'
end;

procedure admresp;
var c:inr;
    n:real;
    d:char;
begin
  nr_adm:=0;
  assign(f,nume_f);
  reset(f);
  repeat
    read(f,c);
    if c.media>=5 then nr_adm:=nr_adm+1
  until eof(f) or (c.media<5) or(nr_adm=nr_loc);
  n:=c.media;
  while (c.media=n) and (n>=5) and (not eof(f))do
  begin
    read(f,c);
    if c.media=n then nr_adm:=nr_adm+1;
  end;
  nr_resp:=filesize(f)-nr_adm;
  clrscr;
```

```
  writeln('nr_adm=',nr_adm);
  writeln('nr_resp=',nr_resp);
  close(f);
  writeln('apasati o tasta ');
  d:=readkey
end;

procedure aplic;
var a:men;
    b:string;
begin
  dialog1;
  textbackground(negru);
  clrscr;
  b:='  CREARE    ADAUGARE    STERGERE ';
  b:=b+'MODIFICARE   SORTARE  ';
  b:=b+' LIST-PROB  LIST-ADMS ';
  a.deschidm(2,1,77,1,rosu,verde,0,11,0,b,false);
  repeat
    if a.selectat then a.salvez;
    case a.citxb of
      1: creare;
      12: adaugare;
      23: stergere;
      34: modificare;
      45: sortare;
      56: listare_p;
      67: begin
            dialog;
            admresp;
            listare_a
          end
    end {case};
    a.restaurezm
  until a.inchis
end;
end.
```

Schema de principiu a aplicației:

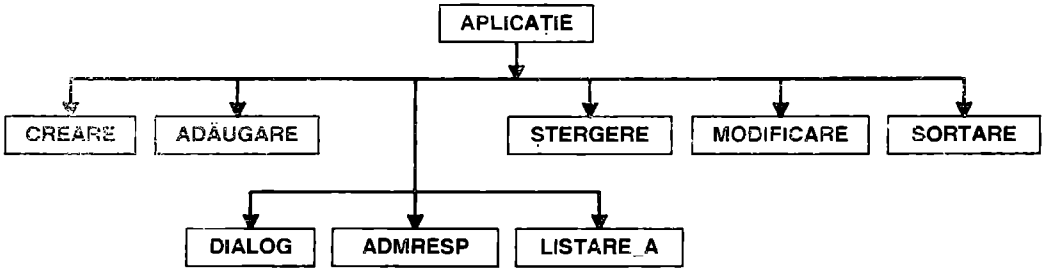


Figura 8.4.1.

8.5. Fișiere fără tip

Fișierele fără tip sunt constituite din blocuri de lungime fixă, motiv pentru care acestea se mai numesc și fișiere cu prelucrare la nivel de bloc. În cadrul blocurilor, informația se scrie direct sub forma în care apare în memoria internă, fără a se converti.

Blocurile se numerotează cu numere cuprinse între 0 și n. Din acest motiv se poate folosi procedura SEEK în forma cunoscută și ea are rolul de a poziționa pointer-ul pe un anume bloc. O variabilă de tip fișier fără tip se declară cu ajutorul cuvântului cheie **FILE**.

Exemplu: a:file;

Asignarea fișierului la suportul extern se face cu ajutorul procedurii **ASSIGN**, exact cum se procedează și în cazul celorlalte tipuri de fișiere.

Deschiderea fișierelor pentru creare se realizează cu ajutorul procedurii **REWRITE**. Forma generală a acestei proceduri este:

REWRITE(var nume fișier,[număr de octeți pentru un bloc])

În cazul în care al doilea parametru este absent, blocul va avea o lungime standard de 128 octeți.

Scrierea blocurilor se face cu ajutorul procedurii **BLOCKWRITE** care are următoarea formă generală:

BLOCKWRITE(nume fișier, nume variabilă în care se face citirea, număr de blocuri care se citesc [,variabila ce reține numărul de blocuri efectiv citite]).

În cazul în care ultimul parametru este absent și se citesc mai puține blocuri decât au fost solicitate, procedura se termină cu eroare de intrare / ieșire.

Programul care urmează creează un fișier fără tip cu n înregistrări.

```
program fft1;
type inreg=  record
               nume:string[16];
               vârsta:integer;
             end;
var  f:file;
     inr:inreg;
     n,i:integer;

begin
  assign(f,'fblk');
  write('n=');
  readln(n);
  rewrite(f,19);
  for i:=1 to n do
  begin
    write('nume=');
    readln(inr.nume);
    write('vârsta=');
    readln(inr.vârsta);
    blockwrite(f,inr,1)
```

```
    end;
    close(f)
end.
```

Deschiderea pentru citire (sau scriere, în caz de actualizare a fișierului) se face cu ajutorul procedurii **RESET** care are forma de apel:

RESET(variabila fișier [,număr de octeți pe care a fost scris blocul]).

Citirea unui bloc se face cu ajutorul procedurii **BLOCKREAD**, în care:

BLOCKREAD(variabila fișier, variabila unde se face citirea, numărul de blocuri care se vor citi[, variabila care reține numărul de blocuri efectiv citite]).

Sfârșitul de fișier se testează cu ajutorul funcției booleene **EOF**, la fel ca și la celelalte tipuri de fișiere.

Programul care urmează citește fișierul care a fost creat anterior.

```
program fft2;
type inreg=  record
               nume:string[16];
               vârsta:integer;
             end;
var f:file;
    inr:inreg;

begin
  assign(f,'fblk');
  reset(f,19);
  while not eof(f) do
  begin
    blockread(f,inr,1);
    writeln(inr.nume, '  ', inr.vârsta)
  end;
  close(f)
end.
```

Nu este obligatoriu să citim un fișier fără tip într-o variabilă de tip record. Blocul se memorează începând cu o anumită variabilă, pe toată lungimea lui (acolo trebuie să se găsească alte variabile în care să se facă memorarea). Un exemplu, în acest sens, îl constituie programul următor (listează același fișier).

```
program fft3;
{$a-}
var f:file;
    nume:string[16];
    vârsta:integer;

begin
  assign(f,'fblk');
  reset(f,19);
  while not eof(f) do
  begin
    blockread(f,nume,1);
    writeln(nume, '  ', vârsta)
```

```
    end;
    close(f)
end.
```

Ne putem întreba care este rezultatul opțiunii de compilare {$a-}. În absența ei, compilatorul rezervă spațiu pentru variabile în memoria internă, după un anumit mecanism. Astfel, pentru o variabilă de tip integer se rezervă doi octeți la o adresă multiplu de 2. Din acest motiv, apar octeți neocupați de variabile, care însă sunt folosiți la citirea blocului. În acest fel, este posibil să nu regăsim rezultatele pe care le așteptăm. Rolul opțiunii de compilare este acela de a comanda compilatorului ca toate variabilele să fie reținute la adrese multiple de 1 (deci nu rămân octeți nerezervați).

Și pentru aceste fișiere poate fi folosită funcția **FILESIZE** la fel ca la cele cu tip. Toate operațiile de actualizare se fac la fel ca la fișierele cu tip, motiv pentru care nu le prezentăm aici.

8.6. Aplicații la capitolul 8

8.6.1. Aplicații ale fișierelor

8.6.1.1. Fișiere text

1. Scrieți un program care creează un fișier text.

Observație. Un fișier text poate fi creat și de la tastatură prin comanda COPY CON, după care se introduc liniile acestuia (liniile se separă prin enter, iar la sfârșit se apasă simultan tastele (CTRL+Z).

2. Se consideră un fișier text cu mai multe înregistrări. Să se afișeze pe monitor (se va ține cont ca după scrierea a 20 de rânduri să se aștepte apăsarea unei taste).

3. Aceeași problemă, numai că fișierul se va copia la imprimantă (pentru o pagină de format A4 se vor scrie 55 de rânduri).

4. Să se creeze un fișier text care conține și înregistrări cu un număr mai mare de 65 de caractere. Fișierul se va găsi pe unitatea a.

5. Utilizând fișierul text de la problema anterioară, să se creeze un alt fișier text în felul următor:

- o înregistrare cu un număr mai mic sau egal cu 65 de caractere se va copia identic în cel de-al doilea fișier;
- o înregistrare cu un număr mai mare de 65 de caractere se va transforma în două sau mai multe înregistrări cu 65 de caractere (excepție, eventual, ultima).

6. Se consideră un fișier text cu mai multe înregistrări. Să se scrie un program care numără aparițiile unui cuvânt. Separatorii pentru cuvinte sunt:

spațiul, punctul, virgula sau sfârșitul unei înregistrări.

7. Se citesc, dintr-un fișier text, n, întreg, și n valori întregi. Să se tipărească minimul din cele n valori.

Observație. În concursurile școlare, datele de test, furnizate de comisie, se găsesc în fișiere text. Programul candidatului trebuie să citească aceste date din fișier.

8. În cazul programelor care listează situații complexe se procedează și astfel:
- situația este scrisă într-un fișier text;
- fișierul text astfel creat se listează atunci când este cazul.

Imaginați o situație care va fi scoasă într-un fișier text (atenție la aliniere).

9. Scrieți un program care listează fișierul text creat la problema anterioară.

Indicații:

7.

```
program maxim;
var a:text;
    n,i,max,v:integer;

begin
  assign(a,'f.dat');
  reset(a);
  read(a,n);
  read(a,max);
  for i:=2 to n do
  begin
    read(a,v);
    if max<v then max:=v
  end;
  writeln('maximul este  ',max);
end.
```

8.6.1.2. Fișiere cu tip

1. Să se creeze un fișier cu înregistrări care au următoarea structură:
- cod material: string[5];
- denumire material: string[20];
- cod magazie: integer;
- cantitate: real;
- unitate de măsură: string[4];
- preț unitar: real.

Citirea câmpurilor unei înregistrări se va face în poziții fixe. Datele referitoare la cantitate și preț unitar vor fi validate din punct de vedere al naturii numerice. (În situația în care o înregistrare nu este validă, calculatorul va emite un sunet iar cursorul se va poziționa pe câmpul incorect.) Se va testa dacă fișierul există sau nu pe suport iar în caz afirmativ se va avertiza operatorul.

2. Să se scrie un program care actualizează fișierul din problema precedentă.

În acest sens, se va deschide un meniu cu trei operații posibile:
- adăugare - se adaugă la sfârșitul fișierului o înregistrare citită după regulile prezentate în problema anterioară;
- ștergere - se identifică înregistrarea care va fi ștearsă după conținutul câmpurilor cod material și cod magazie (în caz că nu este găsită se dă un mesaj de eroare);
- modificare - se modifică conținutul unei înregistrări identificată după cod material și cod magazie (în caz că înregistrarea nu este găsită se dă mesaj de eroare);
- pentru modificare se afișează înregistrarea cu valorile curente ale câmpurilor (cursorul se află poziționat pe valoarea existentă în primul câmp);
- cu ajutorul săgeților (sus, jos) putem deplasa cursorul pe valorile conținute în alte câmpuri;
- câmpurile se pot modifica (ștergerea unui caracter aflat în stânga cursorului, inserarea unuia sau mai multor caractere);
- odată corectată o astfel de înregistrare se apasă ENTER (în situația în care un câmp este incorect, calculatorul va emite un sunet iar cursorul se va găsi asupra acelui câmp).

3. Să se scrie un program care sortează fișierul descris anterior după valorile câmpului cod material.

4. Să se scrie un program care sortează fișierul după cod magazie, iar în cadrul acestuia după cod material (sortare după două chei). În mod practic, fișierul va conține înregistrările magaziei cu cel mai mic cod sortate după codul material, urmate de înregistrările magaziei cu codul imediat superior, sortate după codul materialului etc.

Indicație. Alăturați cele două coduri (întâi codul magazie, apoi codul material) într-o variabilă de tip string. Sortați fișierul după această valoare.

Altfel: realizați o procedură care sortează după cod material înregistrările dintr-un fișier aflate între anumite limite. Inițial se sortează după cod magazie tot fișierul, apoi cu procedura scrisă se sortează înregistrările care au același cod magazie după cod material.

5. Să se scrie un program care scoate la imprimantă situația caracterizată astfel:
- titlul va fi "SITUAȚIA MATERIALELOR PE MAGAZII";
- subtitlul 1 va fi "MAGAZIA...";
- cap tabel (cuprinde informații referitoare la o înregistrare);
- înregistrările cu materialele din prima magazie;
- total valoric pe magazie;
- subtitlul 2 va fi "MAGAZIA..."
- cap tabel;
- înregistrări cu materiale din cea de-a doua magazie;
- total valoric pe magazie;
- total valoric general (suma valorilor pe toate magaziile);

Programul va fi realizat în următoarele condiții:
- se așteaptă ca imprimanta să fie operațională;
- după ce au fost scrise datele referitoare la o magazie se sare la pagină nouă,

se continuă listarea după apăsarea tastei ENTER, numai dacă imprimanta este pregătită de lucru.

6. Scrieți o aplicație complexă care ține evidența elevilor unui liceu (nume, note pe materii și an școlar, etc.). Aplicația va folosi în diverse raportări făcute de liceu către inspectoratele școlare.

Observații. Această aplicație se va realiza de o grupă cu cel puțin cinci elevi. La realizarea aplicației se va consulta secretariatul liceului. Încercați să puneți în practică aplicația realizată. Dacă aceasta nu a fost bine gândită nu se va utiliza. Dacă după toate perfecționările aduse, tot nu se utilizează – v-ați lovit de adevărata problemă a informaticii la noi în țară.

7. Se consideră un fișier cu înregistrări care au o structură la alegerea dumneavoastră. Fișierul este sortat crescător după valorile unui câmp. Se cere să se precizeze dacă o anumită înregistrare se găsește sau nu în fișier. (Generalizare pentru fișiere a problemei căutării.)

8. Se consideră o matrice cu m linii și n coloane. Se presupune că matricea nu încape în memorie. Ca urmare a acestui fapt este reținută într-un fișier în care fiecare înregistrare reține un element a[i,j]. Matricea este reținută pe linii (întâi toate elementele din prima linie, în ordinea coloanelor, apoi cele din linia 2 în ordinea coloanelor ș.a.m.d.). Să se scrie o funcție care returnează elementul a[i,j].

9. Adunați două matrice cu m linii și n coloane care sunt reținute în fișiere ca aceea din problema anterioară.

10. Calculați suma elementelor plasate sub diagonala principală a unei matrice pătratice reținută într-un fișier.

8.6.1.3. Fișiere fără tip

1. Scrieți o procedură care creează un fișier fără tip care are o singură înregistrare și anume o imagine ecran.

Indicație. Se copiază conținutul memoriei video.

2. Scrieți o procedură care reface o imagine citită dintr-un fișier fără tip (creat la problema precedentă).

3. Scrieți o procedură care salvează conținutul unei ferestre într-un fișier fără tip.

4. Scrieți o procedură care reface conținutul unei ferestre salvat prin procedura anterioară.

5. Concepeți un meniu care va fi salvat și restaurat utilizând procedurile de mai sus.

6. O aplicație deosebit de importantă a fișierelor fără tip este aceea că se poate exploata, ca fișier fără tip, orice fișier. Să se scrie un program care copiază un fișier cu un nume care se citește, într-un alt fișier cu un nume citit.

Datorită importanței pe care o are această problemă, programul se prezintă mai jos.

```
program fft;
var a,b: file;
    inr:byte;
    nume,nume1:string;

begin
  write('numele fisierului care se copiaza:');
  readln(nume);
  write('numele noului fisier:');
  readln(nume1);
  assign(a,nume);
  assign(b,nume1);
  reset(a,1);
  rewrite(b,1);
  while not eof(a) do
  begin
    blockread(a,inr,1);
    blockwrite(b,inr,1)
  end;
  close(a);
  close(b)
end.
```

7. Scrieți un program care vizualizează câte 80 de caractere consecutive ale unui fișier. Deplasarea în cadrul fișierului se realizează cu ajutorul săgeților.

Capitolul 9

Unitatea de program DOS

9.1. Prelucrarea datei

Unitatea de program DOS conține 4 proceduri care permit citirea și modificarea datei și orei.

Procedura GETDATE(an, lună, zi, zi_săpt: word) returnează data curentă. Ziua din săptămână este returnată ca un număr între 0 și 6, unde 0 înseamnă duminică.

Procedura GETTIME(oră, minut, secunde, sutimi: word) returnează ora exactă.

Programul care urmează afișează data și ora exactă.

```
program timp;
uses dos;
var an,luna,zi,ora,minut,secunda,zi_sapt,sutimi:word;

begin
  getdate(an,luna,zi,zi_sapt);
  writeln('data curenta');
  writeln('anul ',an);
  writeln('luna ',luna);
  writeln('ziua  ',zi);
  case zi_sapt of
    0: writeln('duminica');
    1: writeln('luni');
    2: writeln('marti');
    3: writeln('miercuri');
    4: writeln('joi');
    5: writeln('vineri');
    6: writeln('sâmbata')
  end {case};
  gettime(ora,minut,secunda,sutimi);
  writeln('ora ',ora);
  writeln('minute ',minut);
  writeln('secunde ',secunda)
end.
```

Procedura SETDATE(an, lună, zi: word) modifică data curentă reținută de sistem.

Procedura SETTIME(oră, minut, secundă, sutimi:word) modifică ora exactă reținută de sistem.

9.2. Căutarea unui fișier

În multe aplicații este necesar să căutăm un fișier. A găsi un fișier înseamnă să găsim calea către el. Funcția FSEARCH are exact acest rol. *Exemplu*: ne propunem să căutăm fișierul TPC.EXE (acesta este un program livrat de firma BORLAND care compilează un fișier text cu extensia .PAS). Funcția FSEARCH are doi parametri formali: numele fișierului care se caută și cataloagele sau subcataloagele în care se face această căutare.

În *exemplul* care urmează se caută fișierul TPC.EXE în cataloagele WS TP și BGI. În cazul în care fișierul este găsit se returnează calea către acesta (de exemplu C:\TP\TPC.EXE), în caz contrar se returnează șirul vid.

```
program caut;

uses dos;

begin
  writeln(fsearch('tpc.exe','c:\ws;c:\tp;c:\bgi'));
end.
```

Câteodată este util să căutăm un fișier în toate căile definite prin PATH în fișierul AUTOEXEC.BAT (aici, se dau căile de căutare separate prin ";"). Funcția GETENV returnează valoarea unei variabile din mediul DOS. Cum valoarea acestei variabile este șirul de căi precizat mai sus, căutarea se face prin utilizarea lor.

```
program caut1;

uses dos;

begin
  writeln(fsearch('tpc.exe',getenv('path')));
end.
```

Observație. În cazul în care fișierul se găsește în mai multe cataloage este returnată numai prima cale în care a fost găsit.

9.3. Execuția unui program

Un program poate apela spre execuție unul sau mai multe programe. Înainte de a prezenta acest lucru, analizăm parametrii de apel ai unui program. În general, apelul unui program executabil se face prin numele lui. Uneori, acesta este urmat de unul sau mai mulți parametri. De exemplu, apelul editorului și compilatorului TURBO PASCAL se poate face și prin cuvântul TURBO urmat de un nume (acesta precizează cum se numește fișierul text care conține programul sursă). Unele programe (de exemplu ARJ) apelate fără nici un parametru se autoprezintă, iar apelate cu unul sau mai mulți parametri realizează ceea ce li se cere prin acești parametri. Parametrii (dacă sunt mai mulți) sunt separați prin spații. Atunci când scriem noi un astfel de program

(care va fi apelat prin parametri) trebuie să putem afla două lucruri:
- care este numărul de parametri cu care a fost apelat programul?
- care sunt parametrii cu care a fost apelat?

Numărul de parametri cu care a fost apelat programul este dat de funcția PARAMCOUNT, funcție de tip WORD. Conținutul unui parametru poate fi găsit cu ajutorul funcției PARAMSTR(i). Aceasta returnează un șir care conține parametrul i.

Programul care urmează deschide un fișier text și citește o primă linie a sa (nu mai mult de 255 de caractere). Numele fișierului este dat ca parametru la apelul acestui program. *Atenție!* Pentru a putea testa acest program, este necesar ca el să fie într-o formă executabilă. Apelați acest program dând ca parametru numele unui text sursă PASCAL.
Observație: cele două funcții prezentate nu aparțin unității DOS.

```
program tparam;

var a:text;
    b:string;

begin
  if paramcount=1 then
  begin
    assign(a,paramstr(1));
    reset(a);
    read(a,b);
    writeln(b);
    close(a);
  end
  else writeln('precizati numele fisierului');
end.
```

Revenim la problema inițială și anume cum putem apela spre execuție, din cadrul unui program, un altul. Acest apel este realizat de procedura EXEC. Această procedură are doi parametri:
- primul este dat de șirul ce conține numele programului care urmează să se execute;
- al doilea reprezintă șirul parametrilor de apel ai programului care se va executa (dacă acest șir este vid se presupune că programul nu este folosit cu parametri de apel).

Procedura EXEC trebuie să fie precedată și urmată de apelul unei alte proceduri numită SWAPVECTORS. Rolul acestei proceduri este mai greu de explicat cu ajutorul cunoștințelor pe care le avem în acest moment. Reținem că ea salvează și restaurează sistemul de întreruperi ale programului care face apelul. Programul care urmează să fie rulat are nevoie de memorie internă unde să fie încărcat. Din acest motiv, se folosește directiva de compilare $M.

Cu ajutorul acestei directive se reduce mărimea stivei pe care o ocupă programul (dimensiunea ei este dată de primul parametru) și se precizează de asemenea minimul și maximul zonei HEAP (ultimii doi parametri). Despre stivă și heap se învață în anul următor, așa că, aici, ne mulțumim să afirmăm că dimensiunea

acestora se reduce pentru a putea fi încărcat programul ce va fi executat.

Programul care urmează apelează spre execuție un altul care se numește CITEL. Acesta nu are parametri de lansare în execuție (al doilea parametru al procedurii EXEC este șirul vid).

```
program exemplu;
{$m 4000,0,$10000}

uses dos;

begin
  swapvectors;
  exec('citel.exe','');
  swapvectors
end.
```

Programul exemplu2, pentru a putea fi testat trebuie să fie executabil. El este apelat cu un parametru care conține numele unui executabil ce va fi lansat în execuție.

```
program exemplu2;
{$m 4000,0,$10000}

uses dos;

begin
  if paramcount=1 then
  begin
    swapvectors;
    exec(paramstr(1),'');
    swapvectors
  end
end.
```

Programul exemplu3 caută compilatorul PASCAL numit TPC. În mod normal, apelul acestuia se face prin comanda TPC urmată de numele unui fișier care conține un text sursă PASCAL și are rolul de a compila textul sursă și de a crea un executabil. Dacă programul TPC este găsit, este lansat în execuție spre a compila un text sursă al cărui nume se citește în variabila de tip string B.

```
program exemplu3;
{$m 4000,0,$10000}

uses dos;
var a,b:string;

begin
  a:=fsearch('tpc.exe',getenv('path'));
  if a<>'' then
  begin
    write('compilam programul :');
    readln(b);
    exec(a,b)
```

```
  end
end.
```

Un program se poate termina cu ajutorul unei proceduri numită HALT. Aceasta întrerupe brusc execuția programului (de multe ori este folosită pentru ieșiri forțate, fapt care încalcă principiile programării structurate). Această procedură are însă și un mare avantaj și anume că poate fixa un cod de retur cu care s-a terminat un program. Pentru a putea realiza această ultimă funcție, se apelează prin HALT(i). De exemplu, putem încheia execuția unui program prin HALT(1), ceea ce înseamnă că programul s-a terminat cu codul de retur 1. Acest cod de retur poate fi exploatat de un program care a apelat spre execuție un altul. De exemplu, în funcție de situație, un program se poate încheia cu codul de retur 0 (dacă nu s-a folosit HALT sau s-a încheiat cu HALT(0) sau cu un alt cod de retur). În funcție de valoarea codului de retur se poate lua în mod automat o decizie sau alta.

Dacă a fost dat spre compilare programului TPC un text care conține erori de sintaxă (compilarea nu a reușit) programul se termină cu un cod de retur diferit de 0, în caz contrar se termină cu codul de retur 0. Funcția DOSEXITCODE (de tip word) citește codul de retur cu care s-a terminat un program.

În final, menționăm faptul că unitatea DOS conține multe alte proceduri și funcții pe care nu le prezentăm aici pentru că depășesc cu mult nivelul acestei clase.

9.4. Probleme propuse

1. Scrieți un program „deșteptător" care citește o oră la care calculatorul va emite un sunet.

2. Scrieți o secvență care efectuează un anumit calcul. Cum putem preciza timpul ei de rulare?

3. Aflați numărul de zile pe care le-ați trăit (data curentă o preluăm din sistem).

4. Completați programul de la jocul CENTRATE (vezi capitolul referitor la subprograme) în așa fel încât partenerul uman să piardă dacă depășește un anumit timp până efectuează o mutare. Timpul scurs va fi prezent tot timpul pe ecran.

5. Scrieți un program care creează și actualizează un fișier cu un nume care figurează ca parametru de apel pentru program.

6. Scrieți un program care copiază un fișier. Numele fișierului sursă și cel al fișierului destinație se dau ca parametri de apel.

7. Scrieți un program care selectează spre execuție alte trei programe. Selecția se va face utilizând un meniu.

8. Scrieți un program care citește un vector. Programul se termină cu codul de retur 0, dacă vectorul are componentele distincte și 1 în caz contrar. Acest program va fi apelat spre execuție de un altul, în mod repetitiv, până când vectorul citit va avea componentele distincte.

Capitolul 10

Grafică pe calculator

10.1. Introducere

În perioada actuală este de neconceput să realizăm programe care se pot utiliza în practică, fără să folosim grafica pe calculator. De multe ori un desen valorează mai mult decât o mie de vorbe. Explicați cuiva cum se ajunge acasă la dumneavoastră și veți observa că înțelege mai ușor acest lucru după un desen.

Limbajul TURBO PASCAL conține o serie de proceduri și funcții care, chiar dacă nu strălucesc din punct de vedere al modului în care funcționează, permit realizarea unor aplicații grafice. Acestea sunt reunite în unitatea GRAPH care se găsește în subcatalogul BGI. În cadrul acestui capitol se vor prezenta o serie de proceduri și funcții, des folosite, cu rol didactic, care se găsesc într-o unitate de program numită UTILG, prezentată integral la sfârșitul acestui capitol.

10.2. Inițializarea modului grafic

Pentru ca o imagine să poată apărea pe ecran, calculatorul trebuie înzestrat cu o placă grafică ce conține memoria video (în care se reține informații asupra imaginii). Există mai multe tipuri de plăci grafice (CGA, EGA, VGA) care diferă prin memorie și alți parametri. Întrucât fabricanții acestor plăci nu au ajuns la o concluzie comună în ceea ce privește folosirea lor, un program care funcționează perfect pe una din ele nu se poate folosi pe o alta. În concluzie, orice limbaj care folosește aceste plăci trebuie să conțină rutine specifice lor. În acest sens, limbajul TURBO PASCAL conține mai multe componente soft care lucrează cu diversele plăci, numite DRIVERE. În funcție de placa grafică găsită, se încarcă un driver sau altul. Acestea se găsesc în fișiere cu extensia .BGI. Astfel, există drivere CGA, VGA etc. Odată ales un driver se alege modul de lucru: în unele moduri de lucru avem o rezoluție mai bună (se reprezintă mai multe puncte pe ecran) dar avem mai puține culori și se pot reține mai puține pagini video (informația referitoare la imaginea grafică conținută de un ecran), în altele avem o rezoluție mai proastă dar putem reprezenta un desen utilizând mai multe culori și putem reține mai multe pagini video. Deci, un mod de lucru este un compromis între numărul de culori, rezoluție și numărul de pagini video. În acest capitol nu vom studia plăci grafice depășite moral cum ar fi cele CGA și EGA. Ne vom concentra atenția în special asupra plăcii VGA, căreia îi

corespunde driverul VGA care poate lucra în mai multe moduri de lucru:
1. modul VGALo cu o rezoluție de 640*200, poate folosi 16 culori la un moment dat și reține 4 pagini video;
2. modul VGAMED cu o rezoluție de 640*350, 16 culori, 2 pagini video;
3. modul VGAHI cu rezoluție 640*480, o singură pagină video, 16 culori.

Selectarea driverului și a modului de lucru se face prin utilizarea procedurii INITGRAPH. Aceasta are trei parametri: gdriver (integer) care conține driverul (codul asociat acestuia), gmode (integer) care conține modul de lucru și o variabilă de tip string care arată calea către unitatea GRAPH. Forma generală a acestei proceduri este: INITGRAPH(gdriver,gmode,'cale'). Primii doi parametri sunt transmiși prin referință. Inițializarea sistemului grafic se poate face în două moduri:

- prin a solicita să se identifice automat placa grafică și corespunzător ei să se încarce un anumit driver și să se selecteze un anumit mod de lucru (în acest caz se alege acel mod de lucru care are cea mai bună rezoluție grafică);
- prin indicarea cu ajutorul primilor doi parametri a unui driver și a unui mod de lucru solicitate de programator (un astfel de program nu se poate executa dacă nu avem placa grafică corespunzătoare);

În cazul primului mod de inițializare se poate folosi procedura INITG care se găsește în unitatea UTILG.

```
procedure initg;
begin
  gdriver:=detect;
  initgraph(gdriver,gmode,'c:\tp\bgi');
  if graphresult<>0 then
  begin
    writeln('tentativa esuata');
    halt
  end
end;
```

Constanta DETECT are valoarea 0 și îi specifică procedurii identificarea automată a driverului și a modului de lucru.

Tentativa de inițializare grafică poate eșua din diverse motive cum ar fi: lipsa unității GRAPH, calea indicată este greșită etc. Pentru a testa dacă inițializarea modului grafic a reușit se folosește funcția întreagă GRAPHRESULT care returnează valoarea 0 în caz afirmativ și o valoare diferită de 0 în caz contrar. În caz de nereușită se oprește forțat programul (halt). În acest caz se renunță la programarea structurată, însă este absurd să scriem întregul program sub clauza ELSE.

În cazul modului 2 de inițializare se poate folosi o secvență ca aceea de mai jos:

```
gdriver:=VGA; gmode:=VGALo;
initgraph(gdriver,gmode,'c:\tp\bgi');
if graphresult<>0 then
begin
```

```
    writeln('tentativa esuata');
    halt
  end;
```

VGA, VGALo sunt constante de tip integer care reprezintă numerele asociate modului și driverului solicitate. Driverele și modurile pot fi adresate prin utilizarea acestor constante.

Odată intrați într-un mod grafic nu se mai poate scrie pe monitor, ca până acum (cu WRITE și WRITELN). Ieșirea din modul grafic se face prin utilizarea procedurii CLOSEGRAPH. În cazul în care nu folosim această procedură, după execuția unui program grafic riscăm dispariția cursorului.

10.3. Culori

Cum putem programa culorile cu care urmează să apară un desen pe monitor? Aceasta este întrebarea la care ne propunem să răspundem în acest paragraf. De la bun început trebuie să precizăm faptul că programarea culorilor ține cont de placa grafică folosită (în concluzie, de driverul folosit) precum și de modul grafic în care se lucrează. Astfel, există moduri grafice monocrome în care avem numai două culori și moduri grafice care dispun de mai multe culori.

Fiind dată o placă grafică, putem folosi memoria ei în mai multe moduri. Astfel (la placa CGA), putem renunța la utilizarea culorilor (de fapt vom utiliza numai două). În acest caz, întrucât pentru fiecare pixel se reține doar dacă este aprins sau stins, în memoria video se pot reprezenta mai mulți biți, deci se obține o rezoluție superioară.
În situația în care se folosesc mai multe culori reprezentăm pe ecran mai puțini pixeli.

În cele ce urmează, vom presupune că dispunem de o placă grafică VGA, caz în care pentru culoarea unui pixel se rețin 4 biți. Rezultă de aici că putem utiliza maximum 16 culori, numerotate între 0 și 15. Nu este cam puțin? Nu ați observat că jocurile folosesc uneori mai multe culori? Dacă le-am folosi, ar fi necesară o memorie video mult mai mare. Pentru a putea reprezenta mai multe culori (dar nu în același timp) se poate schimba setul de culori folosite. Acum intervine noțiunea de paletă de culori. Aceasta este, în esență, mulțimea culorilor care pot fi folosite la un moment dat pentru a reprezenta o imagine grafică. Culorile sunt date prin codul lor și sunt reținute într-un vector. Fiecare componentă a acestui vector reține codul unei anumite culori. În acest fel, conținutul celor 4 biți rezervați pentru culoare este un indice de adresare în paleta de culori (nu culoarea). Astfel, dacă presupunem conținutul celor 4 biți de culoare ai unui pixel de pe ecran ca fiind 0011 (3 în baza 10), culoarea cu care apare efectiv pixelul pe monitor depinde de codul culorii care se găsește în componenta de indice 3 a paletei. Schimbarea paletei de culori atrage modificarea instantanee a culorilor cu care apare un desen pe ecran. O paletă de culori poate fi reținută într-o variabilă de un tip predefinit în unitatea GRAPH și anume tipul PALETTETYPE:

```
type palettetype = record
                     size:byte;
                     colors[0..maxcolors] of shortint
                   end
```

Maxcolors este o constantă definită în graph care are valoarea 15. Câmpul SIZE conține numărul de culori din paletă (în cazul plăcilor VGA are valoarea 16).

Culoarea de fond este întotdeauna culoarea care se găsește în componenta de indice 0 a paletei. În absența unei alte opțiuni, culoarea cu care se face desenul este culoarea al cărei cod se găsește în componenta 15 a paletei. La selecția unui mod grafic se selectează automat o anumită paletă, numită paletă implicită. Aceasta poate fi schimbată. Paleta care dă culorile de pe ecran la un moment dat se numește paletă activă.

Programul care urmează desenează 16 zone dreptunghiulare care sunt colorate cu ajutorul culorilor conținute în paleta implicită (care este și cea activă).

```
program col1;
uses graph,crt;
var gdriver,gmode,i:integer;

begin
  gdriver:=vga;
  gmode:=vgalo;
  initgraph(gdriver,gmode,'c:\tp\bgi');
  if graphresult<>0 then halt;
  for i:=0 to 15 do
  begin
    setfillstyle(1,i);
    bar(0,i*10,200,i*10+10);
  end;
  readln;
end.
```

Cele mai multe programe folosesc numai culori aparținând paletei implicite. În cazul paletei implicite, pentru selecția culorilor se pot folosi constantele de culoare conținute în GRAPH. Astfel, fiecare culoare din paletă are un nume căruia i se atribuie o constantă între 0 și 15 care reprezintă indicele componentei din paletă care reține codul culorii (vezi tabelul).

În cazul în care schimbăm paleta, aceste constante nu mai pot fi folosite.

Culoare	Constanta	Valoare
negru	black	0
albastru	blue	1
verde	green	2
turcoaz	cyan	3
roșu	red	4

Culoare	Constanta	Valoare
violet	magenta	5
maro	brown	6
gri deschis	lightgrey	7
gri închis	darkgrey	8
albastru deschis	lightblue	9
verde deschis	lightgreen	10
turcoaz deschis	lightcyan	11
roșu deschis	lightred	12
violet deschis	lightmagenta	13
galben	yellow	14
alb	white	15

În concluzie, în cazul paletei implicite putem regăsi cu mare ușurință numele culorilor. Stabilirea culorii de fond (dacă se dorește o alta decât culoarea neagră) se face cu ajutorul procedurii SETBKCOLOR(culoare). Parametrul culoare poate avea ca valoare una din constantele definite mai sus sau un număr între 0 și 15. De fapt, această procedură copiază codul culorii care se găsește la indicele specificat în prima componentă a paletei (de indice 0). La indicele culoare codul rămâne neschimbat. Automat, culoarea de fond se schimbă (chiar dacă făcusem un desen). Selectarea culorii cu care se desenează se face prin utilizarea procedurii SETCOLOR(culoare), unde culoarea are aceeași semnificație ca în procedura anterioară. Schimbarea culorii de desen cu SETCOLOR nu influențează desenul făcut anterior, ci numai ce desenăm după apelul acestei proceduri. Programul care urmează desenează un pătrat utilizând linii de culoare albastră pe un fond de culoare albă.

```
program t;
uses utilg,graph;
var i:integer;

procedure desen(culoare:integer);
begin
  setcolor(culoare);
  moveto(10,10);
  lineto(20,10);
  lineto(20,20);
  lineto(10,20);
  lineto(10,10);
```

```
end;

begin
  initg;
  setbkcolor(white);
  desen(blue);
  readln
end.
```

Cum putem schimba paleta și cum putem selecta culorile în cadrul ei? Pentru a putea răspunde acestor întrebări trebuie să analizăm modul de formare a culorilor (codurile lor).

Fiecare culoare se obține ca o combinație între 3 culori fundamentale: roșu, verde, albastru. Culorile fundamentale au și ele mai multe nuanțe, deci și pentru reprezentarea lor sunt necesari mai mulți biți. De exemplu, în modul VGA fiecare culoare fundamentală poate avea un cod între 0 și 63 (în funcție de nuanța ei). Codul final al culorii se obține din combinarea culorilor fundamentale (fiecare cu nuanța ei).

Procedura SETRGBPALETTE atribuie culorii din paleta care are codul C, codul obținut din combinația culorilor fundamentale la gradele de intensitate (nuanțe) n1, n2, n3 (n1 este codul asociat pentru nuanța de roșu, n2 pentru verde și n3 pentru albastru). Această procedură are forma SETRGBPALETTE(C, n1, n2, n3).

Procedura SETPALETTE are rolul de a modifica în paletă culoarea care are indicele de intrare în paletă i, cu un cod de culoare notat C, și are forma generală SETPALETTE(i,C1). Evident, prin utilizarea acestei proceduri se obține o altă paletă. Pentru a ilustra acestea, să considerăm programul următor:

```
program col2;
uses graph;
var gdriver,gmode,i:integer;

procedure schimb_pal;
begin
  for i:=0 to 15 do setpalette(i,i);
  for i:=0 to 15 do setrgbpalette(i,0,0,4*i);
end;

begin
  gdriver:=vga;
  gmode:=vgalo;
  initgraph(gdriver,gmode,'c:\tp\bgi');
  if graphresult<>0 then halt;
  schimb_pal;
  for i:=0 to 15 do
  begin
    setfillstyle(1,i);
    bar(0,i*10,200,i*10+10);
  end;
  readln;
  closegraph;
end.
```

În program, se desenează 16 benzi dreptunghiulare colorate cu diferite nuanțe de albastru. Pentru a obține aceste culori avem nevoie de o paletă care să le conțină. Aceasta se obține cu ajutorul procedurii **schimb_pal**. Cum se procedează? Orice componentă i a paletei inițiale capătă ca valoare culoarea cu codul i. Aceste noi culori nu le folosim în desen ci ca index de adresare pentru procedura SETRGBPALETTE (această procedură adresează culorile din paletă după codurile lor și este convenabil să avem codurile 0,1...15). Ultimul parametru al acestei proceduri conține codul nuanței de albastru.

Procedura GETPALETTE(paleta) are rolul de a copia conținutul paletei curente în cadrul variabilei paletă de tipul PALETTETYPE.

În programul col3 se schimbă paleta la fel ca în programul col2. Se desenează un dreptunghi (culorile le avem în noua paletă). Această figură se „stinge", în sensul că apare într-o culoare de albastru din ce în ce mai închis, până când figura nu mai este vizibilă.

Efectul de „stingere" îl realizăm în felul următor:

- desenăm figura (automat se alege culoarea de index 15), deci albastru intens;
- se citește paleta activă într-o variabilă de tipul palettetype (în acest fel devine accesibil codul culorii care se găsește în cadrul componentei i a paletei active);
- componenta de index 15 va lua pe rând valorile 14, 13, ..., 0, fapt ce atrage modificarea instantanee a culorii desenului.

```
program col3;
uses graph,crt;
var gdriver,gmode,i:integer;
    paleta:palettetype;

procedure desen;
begin
  setfillstyle(1,15);
  bar(100,100,200,200);
end;

procedure schimb_pal;
begin
  for i:=0 to 15 do setpalette(i,i);
  for i:=0 to 15 do setrgbpalette(i,0,0,4*i);
end;

begin
  gdriver:=vga;
  gmode:=vgalo;
  initgraph(gdriver,gmode,'c:\tp\bgi');
  if graphresult<>0 then halt;

  schimb_pal;
  desen;
  getpalette(paleta);
  for i:=14 downto 0 do
```

```
    begin
      delay(40);
      setpalette(15,paleta.colors[i]);
    end;
    closegraph;
  end.
```

Paleta de culori poate fi modificată și prin intermediul procedurii SETALLPALETTE(paleta), unde paleta este o variabilă de tipul PALETTETYPE. Cum poate fi folosită această procedură? Variabila paleta este încărcată de utilizator cu codurile culorilor care se vor folosi. Procedura SETALLPALETTE face ca paleta definită de utilizator să devină activă (deci, în situația în care era desenat ceva, culorile de fond și desen se schimbă instantaneu).

Funcția GETMAXCOLOR este de tip WORD și întoarce cel mai mare indice de intrare în paletă (pentru modul VGA 15).

Funcția GETBKCOLOR este de tip word și întoarce indexul în paletă al culorii de fond active.

Procedura GETDEFAULTPALETTE(paleta) atribuie variabilei paleta (de tipul palettetype) valoarea paletei implicite pentru modul grafic curent. Această procedură poate folosi în refacerea paletei implicite după ce aceasta a fost schimbată (pentru refacere se utilizează și SETALLPALETTE).

Observație: Pentru a obține codurile curente ale culorilor conținute în paleta implicită a modului grafic curent se pot folosi și constantele definite în GRAPH pentru acestea și anume EGABLACK, EGABLUE, EGARED etc.

Procedura CLEARVIEWPORT șterge ecranul cu culoarea de fond. Ea este aplicabilă și ferestrelor, despre care vom discuta mai târziu.

10.4. Coordonate ecran și reprezentarea punctelor

Oricare ar fi modul de lucru ales, un punct se reprezintă printr-un pixel de coordonate x și y (x reprezintă coloana și y linia). Punctul din colțul din stânga sus este de coordonate (0,0). Un punct se poate reprezenta cu ajutorul procedurii PUTPIXEL(x,y,culoare). Funcția GETPIXEL(x,y):word – întoarce indicele culorii din paletă a punctului de coordonate x și y (INTEGER) în cadrul paletei de culori activă. În mod paradoxal, deși acestea ar fi singurele rutine indispensabile programării grafice, ele se utilizează extrem de rar. Coordonata maximă x de pe ecran se poate afla cu ajutorul funcției de tip întreg GETMAXX, iar coordonata maximă Y cu ajutorul funcției GETMAXY. Este bine ca programele de grafică să-și stabilească coordonatele în care desenează, în funcție de valorile returnate de aceste funcții. În acest fel se asigură o relativă independență a programelor față de modul grafic folosit și chiar față de driver.

10.5. Punct curent

Atunci când lucrăm în modul text se poate utiliza cursorul. Orice operație de scriere sau de citire se face în poziția curentă a cursorului. Și în modul grafic dispunem de un mecanism asemănător. Există proceduri grafice care țin cont de poziția unui cursor (invizibil pe ecran), în sensul că pornesc trasarea desenului din acel punct. Coordonatele cursorului imaginar se numesc coordonatele punctului curent. Stabilirea coordonatelor punctului curent se face cu ajutorul procedurii MOVETO(x,y), unde x și y sunt valori de tip INTEGER și reprezintă coordonate ecran.

Nu avem întotdeauna o poziție implicită a punctului curent și nu toate procedurile țin cont de punctul curent. După execuția unora din procedurile grafice, poziția punctului curent se poate modifica; după execuția altora, coordonatele punctului curent rămân neschimbate. Funcțiile de tip întreg GETX și GETY returnează coordonatele x, respectiv y, ale punctului curent.

10.6. Trasarea segmentelor

Două puncte date prin coordonatele lor pot fi unite printr-un segment. Acest segment poate fi trasat prin linii de grosimi diferite, punctate sau nu. Limbajul TURBO PASCAL conține o procedură care are rolul de a fixa modul de trasare a segmentelor (SETLINESTYLE). Forma generală a acesteia este:

SETLINESTYLE(tip_linie,model,grosime).

Toți parametrii sunt de tip WORD. Odată executată, această procedură are efect până la întâlnirea altui apel al ei sau până la sfirșit.

Tipul linie poate avea următoarele valori care sunt recunoscute și prin constantele date:

Tip linie	Valoare	Cod constantă
continuă	0	SOLIDLN
punctată	1	DOTTEDLN
linie punct	2	CENTERLN
linie întreruptă	3	DASHEDLN
definită de utilizator	4	USERBITLN

Parametrul model trebuie să capete valoarea 0, dacă tipul liniei nu este definit de utilizator. În caz contrar, definim printr-o constantă hexa tipul de linie dorit. Definirea se face la nivel de bit (pentru succesiuni de 16 poziții). În

construcția constantei hexa se ține cont de faptul că un bit 0 înseamnă pixel neaprins iar un bit 1 înseamnă pixel aprins. Astfel, dacă se dorește definirea unei linii în care un pixel să fie aprins, un pixel neaprins, se obține șirul de biți 1010101010101010,deci $AAAA.

Parametrul grosime poate avea 2 valori: 1 (NORMWIDTH), pentru linie subțire, și 3 (THICKWIDTH) pentru linie groasă.

În absența acestei proceduri, segmentele se trasează cu linie continuă și subțire.

Următoarele trei proceduri trasează segmente:

1. LINE(x1,y1,x2,y2) - trasează segmente între punctul de coordonate (x1, y1) și punctul de coordonate (x2, y2). Această procedură nu modifică coordonatele punctului curent;
2. LINETO(x,y) - trasează un segment între punctul curent și punctul de coordonate (x,y). După trasare punctul curent are coordonatele (x,y);
3. LINEREL(dx,dy) - trasează un segment între punctul curent și punctul de coordonate (x+dx, y+dy). După trasare coordonatele punctului curent devin (x+dx, y+dy).

Programul care urmează trasează două segmente pentru care a fost precizat stilul.

```
program stil;
uses graph,utilg;

begin
  initg;
  setlinestyle(dottedln,normwidth,1);
  moveto(0,10);
  lineto(200,10);
  setlinestyle(userbitln,$ffff,3);
  moveto(0,20);
  lineto(200,20);
  readln;
  closegraph
end.
```

10.7. Raportul aspect

În general, pixelii nu sunt pătrați, deși există și anumite plăci grafice în care, dacă lucrăm cu anumite moduri de lucru, pixelii îndeplinesc această condiție. Exemplu, pentru o placă VGA, dacă se lucrează în modul VGAHI pixelii sunt pătrați. Dar ce ne facem în celelalte cazuri?

Procedura GETASPECTRATIO ne ajută în acest sens. Forma generală de apel este GETASPECTRATIO(n1,n2), unde n1 și n2 sunt de tip WORD. Procedura furnizează două mărimi, n1 și n2. Definim raportul aspect ca fiind n1/n2 și îl folosim astfel:

- lungimea unui segment, pe orizontală, rămâne nemodificată;

- lungimea unui segment pe verticală se obține ca lungimea dorită înmulțită cu n1/n2.

În programul următor se trasează un pătrat, în două moduri:
- fără a ține cont de raportul aspect;
- ținând cont de acesta.

```
program ra;
uses graph;
var gdriver,gmode:integer;
    n1,n2:word;

begin
  gdriver:=vga;
  gmode:=vgalo;
  initgraph(gdriver,gmode,'c:\tp\bgi');
  if graphresult<>0 then halt;
  moveto(0,0);
  lineto(0,100);
  lineto(100,100);
  lineto(100,0);
  lineto(0,0);
  readln;
  clearviewport;
  getaspectratio(n1,n2);
  moveto(0,0);
  lineto(0,round(100*(n1/n2)));
  lineto(100,round(100*(n1/n2)));
  lineto(100,0);
  lineto(0,0);
  readln;
  closegraph;
end.
```

10.8. Desenarea obiectelor grafice

10.8.1. Desenarea unei linii frânte

Se consideră n puncte de coordonate (x1, y1), (x2, y2), (xn, yn). Coordonatele acestor puncte pot fi reținute într-un vector cu componente POINTTYPE. Acest ultim tip este definit în GRAPH astfel:

```
type pointtype=  record
                   x,y:integer
                 end
```

Procedura DRAWPOLY trasează o linie frântă obținută prin unirea primului punct cu al doilea, al celui de-al doilea cu al treilea ș.a.m.d.

Forma generală a acestei proceduri este: DRAWPOLY(n,vârfuri), unde n este

un parametru de tip word și transmite numărul de puncte ce se vor uni, iar **vârfuri** este un vector de componente POINTTYPE, care reține coordonatele celor n puncte.

Programul următor trasează o linie frântă:

```
program lf;
uses graph,utilg;
var vârfuri:array[1..4] of pointtype;

begin
  initg;
  vârfuri[1].x:=1;
  vârfuri[1].y:=1;
  vârfuri[2].x:=100;
  vârfuri[2].y:=200;
  vârfuri[3].x:=50;
  vârfuri[3].y:=55;
  vârfuri[4].x:=200;
  vârfuri[4].y:=200;
  drawpoly(4,vârfuri);
  readln;
  closegraph
end.
```

În situația în care se dorește trasarea unei linii frânte închise, vectorul VÂRFURI va avea n + 1 componente și ultima componentă va conține coordonatele primului punct. Tipul segmentelor care alcătuiesc linia frântă se stabilește cu ajutorul procedurii SETLINESTYLE.

10.8.2. Desenarea cercurilor, arcelor de cerc și de elipsă

Desenarea unui cerc se face cu ajutorul procedurii CIRCLE(x,y,r), unde (x,y) reprezintă coordonatele centrului iar r reprezintă raza. Toți parametrii sunt de tip WORD.

Desenarea unui arc de cerc se face cu ajutorul procedurii:

ARC(x,y,unghi_pornire, unghi_final,r).

Toți parametrii sunt de tip WORD și au următoarea semnificație: (x,y) coordonatele centrului, unghi inițial și unghi final se dau în grade și precizează limitele arcului, iar r este raza arcului de cerc. Pentru a înțelege limitele în care se trasează cercul, să ne imaginăm cercul trigonometric în care raza vectoare a parcurs în sens trigonometric arcul corespunzător unghiului inițial. Din acest punct (vârful razei) se trasează arcul de cerc până în momentul în care s-a ajuns în punctul corespunzător arcului final. Cu ajutorul acestei proceduri se poate trasa și un cerc, dacă arcul inițial este 0 și cel final este 360 grade.

Desenarea unui arc de elipsă se realizează cu ajutorul procedurii:

ELLIPSE(x,y,unghi_pornire,unghi_final,a,b).

Parametrii, de tip WORD, au semnificația:

- (x,y) coordonatele centrului;
- unghi_pornire, unghi_final au aceeași semnificație ca în cazul arcului de cerc;
- a,b semiaxele elipsei.

În situația în care a este egal cu b se trasează un arc de cerc.

Programul care urmează exemplifică aceste proceduri:

```
program cerc;
uses graph,utilg;

begin
  initg;
  circle(50,50,20);
  arc(100,100,0,270,20);
  ellipse(150,150,0,270,10,30);
  readln;
  closegraph;
end.
```

10.8.3. Desenarea unor obiecte haşurate

Limbajul TURBO PASCAL oferă și posibilitatea desenării unor obiecte haşurate în mod direct, prin utilizarea unor proceduri specifice.

Stabilirea haşurii se face cu ajutorul procedurilor SETFILLSTYLE sau SETFILLPATTERN.

Procedura SETFILLSTYLE (tip_hasura,culoare). Ambii parametri sunt de tip WORD. Valorile parametrului tip_hasura pot fi:

Constanta	Valoare	Descriere interior
EMPTYFILL	0	culoare de fond
SOLIDFILL	1	plin în culoare de umplere
LINEFILL	2	cu linii orizontale
LTSLASHFILL	3	cu linii de forma /// subțiri
SLASHFILL	4	cu linii de forma /// groase
BKSLASHFILL	5	cu linii de forma \\\ groase
LTBKSLASHFILL	6	cu linii de forma \\\ subțiri
HATCHFILL	7	cu linii orizontale și verticale
XHATCHFILL	8	cu linii oblice întrețesute rar
INTERLEAVEFILL	9	cu linii oblice întrețesute des

Constanta	Valoare	Descriere interior
WIDEDOTFILL	10	cu puncte rare
CLOSEDOTFILL	11	cu puncte dese

Parametrul culoare reprezintă culoarea de desen.

În cazul în care nici un astfel de model nu convine, avem posibilitatea de a defini un model propriu. Unitatea GRAPH conține definit tipul următor:

TYPE FILLPATTERNTYPE = array[1..8] of integer;

O variabilă de acest tip definește un pătrat de 8*8 biți din model. Un bit 1 determină aprinderea pixelului, iar pentru un bit 0, pixelul rămâne stins.

Procedura SETFILLPATTERN(model,culoare), unde model este o variabilă de tipul FILLPATTERNTYPE și conține tipul propriu definit de utilizator iar culoarea are semnificația din procedura anterioară.

Procedura BAR(x1,y1,x2,y2) desenează o bară al cărei contur se trasează, cu modelul definit de una din cele două proceduri anterioare. Parametrii, de tip INTEGER, reprezintă coordonatele colțului din stânga sus și dreapta jos.

Procedura BAR3D(x1,y1,x2,y2,adâncime,capac) trasează un paralelipiped pentru care partea frontală este haşurată. Parametrii au următoarea semnificație:

- x1, y1, x2, y2 (de tip INTEGER) coordonatele părții frontale (stânga sus și dreapta jos);
- adâncime;
- capac (boolean), dacă este TRUE se trasează partea superioară, iar dacă are valoarea FALSE, partea superioară nu se trasează.

Programul care urmează exemplifică aceste proceduri:

```
program ob;
uses graph,utilg;

begin
  initg;
  setcolor(green);
  setfillstyle( solidfill,red);
  bar(20,20,100,100);
  bar3d(120,120,200,200,10,true);
  readln;
  closegraph
end.
```

Procedura FILLELLIPSE(x,y,a,b) desenează și haşurează o elipsă. Parametrii au aceeași semnificație ca la procedura ELLIPSE.

Procedura SECTOR(x,y, unghi_start,unghi_f,a,b) desenează și haşurează un sector de elipsă. Parametrii sunt cei de la procedura ELLIPSE.

Procedura PIESLICE(x,y,unghi_start,unghi_f, r) desenează și haşurează un sector de cerc (vezi parametrii procedurii ARC).

Procedura FILLPOLY(n,vârfuri) desenează și haşurează un poligon.

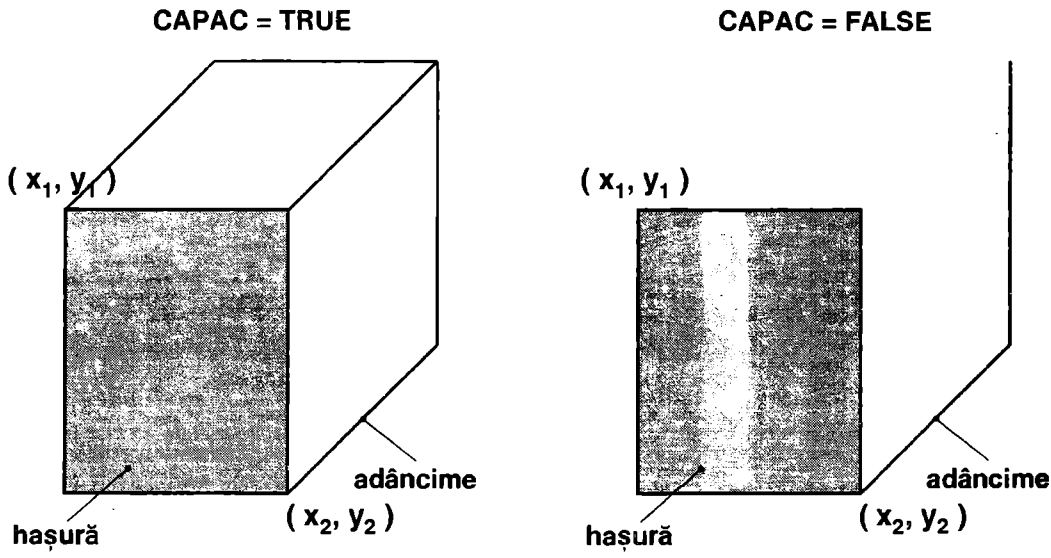


Figura 10.1.

Parametrii sunt aceiaşi cu cei ai procedurii DRAWPOLY.

Programul care urmează arată modul cum se poate realiza grafic o situaţie statistică. Se consideră o firmă care are n magazine. În fiecare lună se raprtează vânzările efectuate în cele n magazine. Acestea se reprezintă procentual faţă de total vânzări firmă, prin sectoare de cerc, aşa cum rezultă din figura următoare.

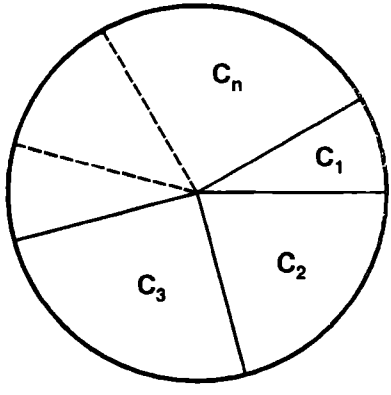


Figura 10.2.

Observaţie: Programul nu scrie texte. Pentru a rezolva această problemă, vezi paragraful următor.

```
program statis1;
uses graph,utilg;
var i,n,s,u_i,u_f:integer;
    realiz:array[1..15] of integer;

begin
  write('n=');
  readln(n);
  s:=0;
  for i:=1 to n do
  begin
    write('realizari magazin ',i,' ');
    readln(realiz[i]);
    s:=s+realiz[i];
  end;
  initg;
  u_i:=0;
  u_f:=0;
  for i:=1 to n do
  begin
    if i=n then u_f:=360
    else u_f:=u_f+round(realiz[i]/s*360);
    setfillstyle(solidfill,i);
    pieslice(getmaxx div 2,getmaxy div 2,u_i,u_f,100);
    u_i:=u_f;
  end;
  readln;
  closegraph
end.
```

10.9. Afișarea textelor

Există mai multe considerente pentru care limbajul a fost dotat cu rutine specifice pentru afișarea textelor în regim grafic. Dintre acestea enumerăm două:

- regimul text este extrem de sărac în ce privește caracterele afișate (după cum am văzut, există un singur tip de caractere);
- există posibilitatea de a crea noi tipuri de caractere.

Întrucât în regimul grafic dispunem de mai multe tipuri de caractere, trebuie să existe o procedură care să ne permită să alegem între ele. Aceasta este procedura SETTEXTSTYLE și are forma următoare:

SETTEXTSTYLE(stil_caracter,direcție,mărime).

Toți parametrii sunt de tip WORD.

Stil_caracter poate avea mai multe valori, după cum rezultă și din tabelul următor:

Nume constantă	Valoare
DEFAULTFONT	0
TRIPLEXFONT	1
SMALLFONT	2
SANSSERIFFONT	3
GOTHICFONT	4

Parametrul direcție specifică direcția de scriere a unui text. Aceasta poate fi orizontală (constanta NORMDIR = 0) sau verticală (constanta VERTDIR = 1). Direcția verticală are sens atunci când tipărim imaginea creată pe ecran (să poată fi tipărită pe o coală așezată normal, adică având baza mică în jos).

Mărime - specifică mărimea caracterelor scrise. Acest parametru trebuie să capete valori între 0 și 10 (0 pentru cel mai mic posibil).

Tipurile de caractere se pot observa în PLANȘA 1, obținută cu ajutorul programului TEXT1.

Textele se afișează colorate în culoarea stabilită cu ajutorul procedurii SETCOLOR (culoarea de desen).

Scrierea textelor se face prin utilizarea a două proceduri: OUTTEXT și OUTTEXTXY. Problema este să stabilim locul de scriere a textelor. Aceasta se realizează cu ajutorul procedurii SETTEXTJUSTIFY(orizontal,vertical). Parametrii, ambii de tip WORD, pot lua valori între 0 și 2. Valorile luate de parametrul orizontal rezultă din următorul tabel:

Constanta	Valoare	Textul se găsește
LEFTTEXT	0	în dreapta punctului curent
CENTERTEXT	1	de o parte și de alta a punctului curent
RIGHTTEXT	2	în stânga punctului curent

Tabelul de mai jos prezintă valorile luate de parametrul vertical:

Constanta	Valoare	Textul se găsește
TOPTEXT	0	deasupra punctului curent
CENTERTEXT	1	de o parte și de alta a punctului curent
BOTTOMTEXT	2	sub punctul curent

În mod evident, parametrul orizontal este luat în considerare când textele se scriu orizontal, iar cel vertical la scrierea verticală a textelor.

În absența acestei proceduri, valorile implicite sunt LEFTTEXT și TOPTEXT.

Procedura OUTTEXT(șir) scrie un text care se găsește în variabila șir de tip STRING. Scrierea textului se face ținând cont de punctul curent, așa cum a fost arătat mai sus.

Procedura OUTTEXTXY(x,y,șir) face ca punctul curent să aibă coordonatele x și y, după care scrie textul ca și procedura prezentată mai sus.

Iată programul text1 care listează tipurile de caractere:

```
program text1;
uses graph,utilg;
var i,j:integer;

begin
  initg;
  setcolor(red);
  for i:=0 to 4 do
  begin
    settextstyle(i,horizdir,6);
    outtextxy(0,i*60,'aAbBcCdD123');
  end;
  readln;
  closegraph
end.
```

Programul care urmează scrie de două ori cuvântul text, orizontal și vertical, în centrul ecranului (cuvintele se intersectează).

```
program text2;
uses graph,utilg;

begin
  initg;
  setcolor(yellow);
  settextstyle(triplexfont,horizdir,10);
  settextjustify(centertext,bottomtext);
  outtextxy(getmaxx div 2, getmaxy div 2, 'text');
  settextstyle(triplexfont,vertdir,10);
  outtextxy(getmaxx div 2, getmaxy div 2, 'text');
  readln;
  closegraph
end.
```

Aplicație:

Se consideră o firmă care dispune de n (n mai mic sau egal cu 8) magazine. Fiecare magazin raportează lunar cifra vânzărilor în milioane lei (între 0 și 999). Se cere să se reprezinte grafic vânzările efectuate în cele n magazine (reprezentări prin paralelipipede de înălțime proporțională cu vânzările efectuate).

Pentru rezolvarea problemei propuse procedăm astfel:

- orice paralelipiped va avea lățimea feței frontale de 50;
- un paralelipiped de înălțime maximă va avea y cuprins între getmaxy-50 și 50;
- înălțimea unui paralelipiped oarecare este exprimată proporțional față de înălțimea paralelipipedului de înălțime maximă (corespunzător magazinului cu vânzări maxime).

Analizați programul de mai jos și justificați formulele folosite.

```
program text3;
uses graph,utilg;
var realiz:array[1..8] of integer;
    i,n,max,x1,y1,x2,y2:integer;
    sir:string;

begin
  write('n=');
  readln(n);
  for i:=1 to n do
  begin
    write('realizari[',i,']=');
    readln(realiz[i]);
  end;
  max:=realiz[1];
  for i:=2 to n do
  if max<realiz[i] then max:=realiz[i];
  initg;
  setcolor(yellow);
  settextstyle(sansseriffont,horizdir,0);
  setfillstyle(hatchfill,red);
  outtextxy(150,0,'Situatia vânzarilor');
  x1:=0; x2:=50; y2:=getmaxy-50;
  for i:=1 to n do
  begin
    y1:=getmaxy-50-round((getmaxy-100)*(realiz[i]/max));
    bar3d(x1,y1,x2,y2,10,true);
    str(realiz[i],sir);
    outtextxy(x1,getmaxy-40,sir);
    x1:=x1+80; x2:=x2+80;
  end;
  readln;
  closegraph
end.
```

Șirul de octeți necesar reprezentării unui caracter se numește font. Caracterele tipului DEFAULTFONT sunt reprezentate în memorie pe câte 8 octeți. Pentru a genera un astfel de caracter se procedează astfel:

- se consideră o matrice binară (numai cu 1 sau 0) de 8*8, corespunzătoare unui pătrățel de câte 8*8 pixeli în care se reprezintă un astfel de caracter. Pentru orice element 1 din matrice, pixelul corespunzător se consideră aprins (are culoarea stabilită de SETCOLOR) iar pentru orice element 0 pixelul se consideră stins (are culoarea fondului). Pentru fiecare linie a matricei se obține un octet care reprezintă un număr. În acest fel se obține șirul celor 8

octeți. Un astfel de font este de tip raster. Dacă am încerca să deformăm un astfel de caracter (să înmulțim cu două numere diferite înălțimea și lățimea caracterului) imaginea rezultată ar fi de slabă calitate.

Celelalte tipuri de caractere au fonturi de tip vectorial. Ce este un font de tip vectorial? Fiecare caracter se reprezintă în memorie prin mai multe perechi de octeți (nu există un număr fix de perechi). Fiecare pereche de octeți cuprinde abscisa și ordonata unui punct. Desenul caracterelor se obține unind punctele ale căror coordonate sunt date de perechile de octeți (de fapt, ultimul bit al octetului care reprezintă ordonata unui punct are valoarea 1 dacă nu se face trasare din punctul anterior la acesta și 0 în caz contrar). Caracterele generate prin fonturi vectoriale se pot deforma cu multă ușurință (se obțin tot puncte care sunt unite prin segmente).

10.10. Tehnici de animație

În acest paragraf ne propunem să răspundem la următoarea întrebare: cum facem o figură să se miște pe ecran? Pentru rezolvarea acestei probleme există mai multe tehnici care vor fi prezentate în continuare, dar toate au același principiu de execuție și anume:

- se desenează figura;
- se așteaptă o perioadă de timp;
- se șterge figura și se desenează într-o altă poziție.

Pentru ca figura să pară că se mișcă în mod real, un rol fundamental îl are timpul de așteptare (timpul în care figura rămâne pe ecran până a se șterge). Acesta se reglează de la caz la caz cu ajutorul procedurii DELAY.

În principal, există trei tehnici de realizare a animației. Ele diferă prin procedurile pe care le utilizează și prin viteza de execuție. Pentru exemplificarea lor vom folosi o singură problemă și anume de a mișca un pătrat pe orizontală.

Tehnica 1

Operațiile de desenare și ștergere a figurii se fac cu ajutorul procedurii SETWRITEMODE, pe care o vom prezenta în continuare.

Forma generală a acestei proceduri este SETWRITEMODE(valoare întreagă). Vom folosi această procedură cu un singur parametru și anume XORPUT (valoarea 1 predefinită în unitatea GRAPH). În concluzie, vom pune SETWRITEMODE(XORPUT).

După apelul acestei proceduri procedăm astfel:

- apelăm o procedură care desenează o figură (aceasta va fi vizibilă pe ecran);
- așteptăm o perioadă de timp;
- apelăm din nou procedura care realizează desenul, exact în aceeași poziție (la acest apel desenul va dispărea, pentru că, de fapt, desenul se realizează acum utilizând culoarea fondului);

- apelăm procedura care realizează desenul într-o altă poziție (acesta va deveni vizibil, pentru că se realizează utilizând culoarea curentă);
- așteptăm o perioadă de timp;
- procedeul se repetă până când figura a ajuns în poziția dorită.

Programul care urmează utilizează această tehnică pentru a plimba pătratul pe o distanță de 100 de pixeli.

```
program a1;
uses  graph,utilg,crt;
var i:integer;

procedure desen(x,y,l:integer);
begin
  moveto(x,y);
  lineto(x+l,y);
  lineto(x+l,y+l);
  lineto(x,y+l);
  lineto(x,y)
end;

begin
  initg;
  setwritemode(xorput);
  for i:=1 to 100 do
  begin
    desen(i,1,20);
    delay(100);
    desen(i,1,20)
  end;
  closegraph
end.
```

Tehnica 2

Pentru a înțelege această tehnică, trebuie să prezentăm în prealabil câteva proceduri.

Orice imagine care se află pe ecran poate fi salvată în memoria internă. Să fim mai expliciți. Există posibilitatea ca anumite informații (cum ar fi cele care permit vizualizarea unei imagini) să fie salvate în memoria internă, alocând spațiul necesar pentru aceasta în timpul execuției programului. O astfel de alocare a memoriei poartă numele de alocare dinamică și se va studia în detaliu în clasa a zecea. Totuși, acum ne sunt necesare câteva cunoștințe minimale.

În programul care urmează, am reținut o variabilă a de un tip încă necunoscut, numit POINTER. O astfel de variabilă are posibilitatea de a reține o adresă de memorie (atenție, nu faceți confuzie între adresa unei zone de memorie și conținutul ei – este ca și cum am face confuzie între adresa unei persoane și persoana propriu-zisă). Alocarea spațiului în memorie (un număr de octeți consecutivi la o anumită adresă) se face cu ajutorul procedurii GETMEM care are forma următoare:

GETMEM(variabila de tip POINTER, număr octeți).

Această procedură, după ce rezervă în memorie numărul cerut de octeți, încarcă adresa primului octet în variabila de tip POINTER.

Cum salvăm imaginea? De fapt, se salvează imaginea conținută într-un dreptunghi pentru care se cunosc coordonatele colțurilor din stânga sus și dreapta jos. De unde știm câți octeți sunt necesari pentru a salva o imagine? Aici, ne este de mare folos funcția IMAGESIZE. Această funcție este de tip WORD și returnează numărul de octeți necesari pentru salvarea unei imagini în cazul când dispunem de acest spațiu, în caz contrar se returnează valoarea 0.

Până acum știm cum putem afla numărul de octeți necesari memorării imaginii, cum să rezervăm spațiul necesar memorării imaginii, dar nu știm cum salvăm efectiv imaginea (octeții corespunzători ei aflați în memoria video). Pentru a putea realiza aceasta, folosim procedura GETIMAGE care are forma următoare:

GETIMAGE(x1,y1,x2,y2,a^).

Parametrii au următoarea semnificație:
- x1,y1,x2,y2 - coordonatele colțului din stânga sus și dreapta jos ale dreptunghiului care conține imaginea;
- a^ - spațiul în care se face salvarea. Referitor la ultimul parametru, facem următoarele precizări:
- a este variabilă de tip POINTER;
- când scriem a, ne referim la adresa unde s-a făcut rezervarea;
- când scriem a^ ne referim la conținutul de la adresa care se găsește în variabila a.

În concluzie, procedura GETIMAGE are rolul de a salva o imagine în memorie. Operația inversă, de a reface imaginea prin aducerea octeților de informație în memoria video, se face prin utilizarea procedurii PUTIMAGE. Aceasta are forma generală:

PUTIMAGE(x1,y1,a^,xorput).

Parametrii au următoarea semnificație:
- x1,y1 coordonatele colțului din stânga sus al dreptunghiului care conține imaginea (nu este obligatoriu să refacem imaginea în poziția pe care a avut-o înainte de salvare, acest fapt permite și animația);
- a^ - conținutul zonei de memorie a cărei adresă se găsește în a;
- parametrul xorput, prezentat și la setwritemode și care are rolul ca odată figura să fie vizibilă, odată nu (se desenează utilizând culoarea de fond). Acum cunoaștem cum salvăm și cum refacem o informație, dar problema era să realizăm animația. Utilizând tehnica salvării și refacerii imaginii, se procedează astfel:
- se desenează figura într-o primă poziție;
- se calculează numărul de octeți necesari salvării ei în memoria internă;
- se salvează imaginea;
- se așteaptă un interval de timp;
- se reface imaginea dar cu culoarea fondului, ceea ce duce de fapt la ștergerea ei;

- repetitiv se procedează astfel:
  - se reface imaginea într-o poziție nouă;
  - se așteaptă un interval de timp;
  - se reface imaginea în aceeași poziție dar, datorită parametrului xorput, de fapt se șterge;
  - procedeul continuă până când s-a ajuns în poziția finală.

Această tehnică de animație este superioară primei tehnici, datorită faptului că este mai rapidă operația de aducere din memorie pe ecran a unei imagini, decât desenarea ei. Acest fapt constituie un mare avantaj în cazul imaginilor complexe.

```
program a2;
uses  graph,utilg,crt;
var i,dim:integer;
        a:pointer;

procedure desen(x,y,l:integer);
begin
  moveto(x,y);
  lineto(x+l,y);
  lineto(x+l,y+l);
  lineto(x,y+l);
  lineto(x,y);
end;

begin
  initg;
  desen(1,1,10);
  dim:=imagesize(1,1,10,10);
  getmem(a,dim);
  getimage(1,1,11,11,a^);
  delay(100);
  putimage(1,1,a^,xorput);
  for i:=2 to 100 do
  begin
    putimage(i,1,a^,xorput);
    delay(100);
    putimage(i,1,a^,xorput)
  end;
  closegraph
end.
```

Tehnica 3

Această tehnică nu este aplicabilă în general, ci numai în cazul în care se lucrează într-un mod grafic, cu un driver grafic ce permite utilizarea mai multor pagini video. O pagină video este memoria necesară (existentă fizic pe placa grafică) ce permite reținerea unei imagini.

Un exemplu în care putem folosi această tehnică este atunci când folosim driverul VGA și lucrăm în modul VGALo, caz în care dispunem de 4 pagini video. Apar două noțiuni noi și anume pagina vizualizată și pagina activă.

Pagina vizualizată este aceea care se vede pe ecran, iar cea activă este cea în care acționează procedurile grafice. Dacă avem o singură pagină video, cele două noțiuni coincid. În cazul în care se lucrează cu mai multe pagini video, una poate fi vizualizată și alta poate fi activă. Stabilirea paginii care se vizualizează se face cu ajutorul procedurii SETVISUALPAGE(nr), unde nr reprezintă numărul paginii. Facem precizarea că paginile se numerotează începând cu 0 (astfel, dacă dispunem de 4 pagini video, acestea sunt numerotate între 0 și 3). Stabilirea paginii active se face cu procedura SETACTIVEPAGE(nr), unde parametrul nr este de tip integer și are semnificația de număr al paginii.

Să analizăm exemplul următor:

- se vizualizează pagina 0 (fără altă precizare, aceasta este și pagina activă);
- se desenează un segment de dreaptă, care este vizibil pe ecran;
- se așteaptă citirea unei taste spre a ne convinge de cele prezentate;
- pagină activă devine pagina 1;
- în pagina activă se desenează un alt segment de dreaptă (acesta nu este vizibil);
- se așteaptă citirea unei taste spre a ne convinge de aceasta;
- se vizualizează pagina 1, moment în care apare pe ecran noul desen.

```
program tp1;
uses graph,utilg,crt;
var gdriver,gmode:integer;

begin
  gdriver:=vga;
  gmode:=vgalo;
  initgraph(gdriver,gmode,'c:/tp/bgi');
  if graphresult<>0 then halt;
  setvisualpage(0);
  moveto(1,1);
  lineto(getmaxx,getmaxy);
  readln;
  setactivepage(1);
  moveto(1,1);
  lineto(100,100);
  readln;
  setvisualpage(1);
  readln;
end.
```

Cum putem utiliza toate acestea spre a putea realiza animația?

Ideea de bază este următoarea: atât timp cât este vizualizată o pagină, într-o alta se realizează desenul în noua poziție. Vă imaginați cât timp se câștigă?

Să analizăm programul următor:

- se vizualizează pagina 0 în care se desenează pătratul;
- pagina 1 devine activă și aici se desenează pătratul în poziția următoare;
- repetitiv se procedează astfel:
  - se așteaptă o perioadă de timp;
  - se vizualizează pagina 1;

- pagină activă devine pagina 0;
- aici, se șterge desenul din poziția curentă și se reface acesta în poziția următoare celei din pagina vizualizată (procedura mișcare);

- se așteaptă o perioadă de timp;
- se vizualizează pagina 0;
- pagină activă devine pagina 1;
- aici, se șterge desenul și se redesenează în poziția următoare celei din pagina vizualizată;
- procedeul continuă până când se ajunge cu figura în poziția finală.

```
program tpl1;
uses graph,crt;
var gdriver,gmode,i,nrp:integer;

procedure desen(x,y,l:integer);
begin
  moveto(x,y);
  lineto(x+l,y);
  lineto(x+l,y+l);
  lineto(x,y+l);
  lineto(x,y)
end;

procedure miscare(i:integer);
begin
  desen(i,1,10);
  desen(i+2,1,10)
end;

begin
  gdriver:=vga;
  gmode:=vgalo;
  initgraph(gdriver,gmode,'c:/tp/bgi');
  if graphresult<>0 then halt;
  setwritemode(xorput);
  setvisualpage(0);
  desen(1,1,10);
  setactivepage(1);
  desen(2,1,10);
  nrp:=1;i:=1;

  repeat delay(100);
    setvisualpage(nrp);
    setactivepage(1-nrp);
    miscare(i);
    nrp:=1-nrp;
    i:=i+1
  until i>100;
end.
```

10.11. Fișiere imagine

În diverse aplicații apare problema salvării imaginilor sub formă de fișiere, iar când este cazul, aceste imagini se refac. Din păcate, limbajul nu are proceduri care să rezolve această problemă. În prezentul paragraf propunem o modalitate simplă, care rezolvă cele propuse.

Unitatea UTILG conține două proceduri SCRIMG și CITIMG.

Procedura SCRIMG salvează o imagine. Pentru aceasta folosește două fișiere, și anume: unul text care conține dimensiunea imaginii salvate și unul fără tip care conține imaginea propriu-zisă. Pentru salvarea imaginii, procedăm astfel:

- se calculează dimensiunea imaginii ce va fi salvată (procedura imagesize) și aceasta se scrie într-un fișier text;
- se salvează în memorie imaginea (procedurile getmem și getimage);
- de aici, imaginea este scrisă într-un fișier fără tip.

Parametrii acestei proceduri sunt:

- x1, y1, x2, y2: coordonatele colțurilor din stânga sus și dreapta jos ale dreptunghiului ce conține imaginea;
- fdim, fimg: parametri de tip **string** care conțin numele fișierelor text și al celui fără tip;
- cod parametru de tip byte care se testează la revenirea din procedură (dacă salvarea a reușit are valoarea 0, în caz contrar are valoarea 1).

```
procedure scrimg(x1,y1,x2,y2:integer;fdim,fimg:string;var cod:byte);
var a:pointer;
    dim:integer;
    f1:text;
    f2:file;

begin
  assign(f1,fdim);
  assign(f2,fimg);
  dim:=imagesize(x1,y1,x2,y2);
  if dim<>0 then
  begin
    getmem(a,dim);
    getimage(x1,y1,x2,y2,a^);
    rewrite(f1);
    rewrite(f2,dim);
    write(f1,dim);
    blockwrite(f2,a^,1);
    close(f1);
    close(f2);
    cod:=0
  end
  else cod:=1
end;
```

Procedura CITIMG are rolul de a reface imaginea conținută în fișiere. Pentru a realiza acest lucru, se procedează astfel:

- se citește dimensiunea din fișierul text;
- se rezervă în memorie spațiu pentru citirea imaginii (procedura getmem);

- se citește în memorie imaginea din fișierul fără tip (procedura blockread);
- se vizualizează imaginea (procedura putimage).

Parametrii acestei proceduri sunt următorii:

- x,y - coordonatele colțului din stânga sus al dreptunghiului care va conține imaginea;
- fdim, fimg - parametrii de tip string care conțin numele celor două fișiere.

```
procedure citimg(x,y:integer;fdim,fimg:string);
var f1:text;
    f2:file;
    dim:integer;
    a:pointer;

begin
  assign(f1,fdim);
  assign(f2,fimg);
  reset(f1);
  read(f1,dim);
  reset(f2,dim);
  getmem(a,dim);
  blockread(f2,a^,1);
  putimage(x,y,a^,xorput);
  close(f1);
  close(f2);
end;
```

Următoarele două programe testează cele prezentate.
Programul IMSALV desenează un cerc și îl salvează.

```
program imsalv;
uses graph,utilg;
var cod:byte;

begin
  initg;
  circle(10,10,10);
  scrimg(0,0,20,20,'dimg','img',cod);
  closegraph;
  if cod=0   then writeln('imagine salvata')
             else writeln('tentativa de salvare esuata')
end.
```

Programul IMCIT reface imaginea salvată de programul anterior (se observă faptul că imaginea este refăcută într-o altă poziție, pentru că așa am solicitat).

```
program imcit;
uses graph,utilg;

begin
  initg;
  citimg(100,100,'dimg','img');
  readln;
  closegraph;
end.
```

10.12. Ferestre grafice

În studiul unității CRT ne-am obișnuit să lucrăm cu ferestre în care se puteau reprezenta caractere. Se poate lucra cu ferestre și în grafică, însă procedurile puse la dispoziție de unitatea de program GRAPH, pentru această problemă, sunt mult mai modeste.

Deschiderea unei ferestre se face prin procedura SETVIEWPORT. Forma generală de apel a acestei proceduri este următoarea:

SETVIEWPORT(x1,y1,x2,y2,clip);

- (x1, y1) reprezintă coordonatele colțului din stânga sus al ferestrei;
- (x2, y2) reprezintă coordonatele colțului din dreapta jos al ferestrei;
- parametrul clip este boolean, dacă are valoarea **true** nu este permisă scrierea în afara ferestrei (la tentativa de desenare a unui segment care intersectează fereastra, se trasează numai porțiunea din interiorul ferestrei), în caz contrar se poate scrie și în afara ei.

Odată deschisă o fereastră, orice procedură grafică ce folosește coordonate de puncte, va utiliza coordonatele relativ la fereastra nou creată. Acest aspect permite scrierea unor programe independente de fereastra folosită.

Culoarea cu care se desenează într-o fereastră se stabilește, cum este și normal, cu ajutorul procedurii setcolor. Dar culoarea fondului? În cazul utilizării procedurii setbkcolor, nu numai fereastra capătă culoarea cerută, ci întreg ecranul. Nici măcar nu există o altă procedură care să ofere această posibilitate. De asemenea, chiar dacă avem deschisă o fereastră, funcțiile getmaxx și getmaxy sunt relative la întreg ecranul.

Programul care urmează afișează cele mai mari coordonate de afișare (care sunt relative la întregul ecran, deși s-a deschis o fereastră).

```
program t1;
uses graph,utilg;
var a,b:integer;

begin
  initg;
  setviewport(1,1,5,5,true);
  a:=getmaxx; b:=getmaxy;
  closegraph;
  writeln(a,' ',b);
end.
```

Datorită acestor neajunsuri, este necesar să construim un obiect numit fers (fereastră obiect), care elimină o parte din ele. Acest obiect are următoarele date:

- x1, y1, x2, y2 - coordonatele ferestrei;
- cf, cc - culorile de fond și de desen;
- maxx, maxy - coordonatele maxime care pot fi afișate în fereastră;
- lung - dimensiunea în octeți necesară salvării ferestrei;
- clip - dacă fereastra este cu tăiere sau nu;
- adresa - reține adresa în cazul în care fereastra a fost salvată.

Acest obiect conține și o serie de metode care vor fi prezentate în continuare.

Metoda INIT rezolvă următoarele:
- deschide o fereastră;
- reține pentru ea culorile de fond, de desen și parametrul de tăiere;
- calculează maxx și maxy;
- atribuie ferestrei culoarea de fond (de fapt desenează porțiunea corespunzătoare ei, a ferestrei implicite, cu ajutorul procedurii BAR, după care o deschide);
- stabilește culoarea de desen (SETCOLOR).

Metodele CITMAXX SI CITMAXY sunt funcții de tip integer asemănătoare funcțiilor GETMAXX și GETMAXY, care au rolul de a furniza utilizatorului conținutul variabilelor MAXX și MAXY (coordonate maxime de afișare pe ecran).

Metoda SALVEZ are rolul de a salva o fereastră în memorie (de fapt, imaginea asociată ei). Pentru a realiza aceasta, se procedează astfel:
- se redeschide fereastra mare (cea implicită);
- se calculează dimensiunea necesară salvării imaginii ferestrei;
- se alocă spațiu în memorie și se salvează fereastra (în cazul unei tentative nereușite, variabila cod capătă valoarea 0).

Metoda RESTAUREZ are rolul de a restaura o fereastră salvată (să reapară imaginea ei pe ecran). După restaurare, se eliberează spațiul de memorie rezervat imaginii.

Metoda INCHID are rolul de a închide fereastra.

Iată unitatea de program:

```
unit tfer;

interface
uses utilg,graph;
type fers =  object
               x1,y1,x2,y2,cf,cc,maxx,maxy,lung:integer;
               clip:boolean;
               adresa:pointer;
               procedure init(xa,ya,xb,yb,cfond,codc: integer;
                              tai:boolean);
               function citmaxx:integer;
               function citmaxy:integer;
               procedure salvez(cod:integer);
               procedure restaurez;
               procedure inchid;
             end;

implementation

procedure fers.init;
var i:integer;
begin
  x1:=xa;
  y1:=ya;
```

```
    x2:=xb;
    y2:=yb;
    cc:=codc;
    cf:=cfond;
    clip:=tai;
    maxx:=x2-x1;
    maxy:=y2-y1;
    setfillstyle(1,cf);
    bar(x1,y1,x2,y2);
    setcolor(cc);
    setviewport(x1,y1,x2,y2,clip);
  end;

  function fers.citmaxx:integer;
  begin
    citmaxx:=maxx;
  end;

  function fers.citmaxy:integer;
  begin
    citmaxy:=maxy;
  end;

  procedure fers.salvez;
  begin
    setviewport(0,0,getmaxx,getmaxy,false);
    cod:=imagesize(x1,y1,x2,y2);
    if cod<>0 then
    begin
      lung:=cod;
      getmem(adresa,lung);
      getimage(x1,y1,x2,y2,adresa^);
      clearviewport
    end
  end;

  procedure fers.restaurez;
  begin
    setviewport(0,0,getmaxx,getmaxy,false);
    putimage(x1,y1,adresa^,0);
    setviewport(x1,y1,x2,y2,true);
    setcolor(cc);
    freemem(adresa,lung);
  end;

  procedure fers.inchid;
  begin
    clearviewport;
    setviewport(0,0,getmaxx,getmaxy,false);
  end;
```

Programul care urmează testează obiectul FERS. Acesta realizează următoarele:

- deschide o fereastră în care trasează un segment;

- salvează fereastra;
- deschide o a doua fereastră în care trasează un segment;
- restaurează prima fereastră în care trasează un nou segment;
- închide cele două ferestre.

```
program testf;
uses graph,utilg,tfer;
var a,b:fers;
    cod:integer;

begin
  initg;
  a.init(30,100,300,200,red,green,true);
  moveto(0,0);
  lineto(a.citmaxx,a.citmaxy);
  readln;
  a.salvez(cod);
  readln;
  b.init(100,100,200,200,yellow,black,true);
  moveto(0,b.citmaxy);
  lineto(b.citmaxx,0);
  readln;
  b.inchid;
  a.restaurez;
  readln;
  moveto(10,10);
  lineto(50,50);
  readln;
  a.inchid;
  readln;
  closegraph
end.
```

10.13. Elemente de grafică 2D

10.13.1. Desenarea graficului unei funcții

Se consideră funcția f definită pe intervalul [a,b] cu valori în mulțimea numerelor reale. Se cere să se reprezinte grafic funcția f.

Reamintim definiția graficului unei funcții f, definiție cunoscută din matematică: $Gf=\{(x,y)|a<x<b \ \& \ y=f(x)\}$. Evident, graficul unei funcții definită pe un interval va conține o infinitate de puncte (x,f(x)). Numărul de puncte care pot fi reprezentate pe ecran este finit, motiv pentru care nu le putem reprezenta pe toate. O funcție poate fi reprezentată printr-un număr maxim de puncte egal cu numărul de pixeli conținuți de un rând al ecranului (getmaxx+1).

Programul care urmează reprezintă grafic funcția $f=\sin(x)$ pe intervalul $[-\pi,\pi]$.

```
program graf1;
uses utilg,graph;
var i:integer;

begin
  initg;
  moveto(0,round(getmaxy/2));
  lineto(getmaxx,round(getmaxy/2));
  moveto(round(getmaxx/2),0);
  lineto(round(getmaxx/2),getmaxy);
  moveto(0,round(getmaxy/2-sin(-pi)*getmaxy/2));
  for i:=1 to getmaxx do
    lineto(i,round(getmaxy/2-sin(-pi+2*pi/getmaxx*i)*getmaxy/2))
end.
```

Cum s-a procedat?

În primul rând au fost trasate axele OX și OY. Întrucât domeniul de definiție al funcției considerate este simetric ($[-\pi,\pi]$), axa OY se trasează în mijlocul ecranului. Un punct aparținând graficului funcției are două coordonate, x și y. Indiferent de funcție și de domeniul ei de definiție, x va lua pe rând valorile 0, 1,...,getmaxx. Care sunt valorile luate de y? În mod normal y = sin(x). Dar x ia valorile arătate, care nu au nici o legătură cu intervalul $[-\pi,\pi]$. Calculăm:

y = sin(−pi + 2 * pi/getmaxx * i),

unde i ia valori între 0 și getmaxx. Când i ia valoarea 0 se calculează y = sin(−pi). Când i ia valoarea getmaxx se calculează sin(pi). În esență, se face un calcul de forma sin(−pi + r * i), unde r = 2 * pi/getmaxx. Nici acest calcul nu permite reprezentarea grafică dorită. Se știe că funcția sinus ia valori în intervalul [−1,1]. Deci y va lua valori aparținând acestui interval. Ordonata unui punct de pe ecran trebuie să fie cuprinsă între 0 și getmaxy, altfel graficul apare pe ecran ca o linie. Calculăm:

y = sin(−pi + 2 * pi/getmaxx * i) * getmaxy/2 .

În acest fel, valorile y se vor găsi în intervalul [−getmaxy/2,getmaxy/2]. Cele două valori extreme au fost obținute ținând cont că sinusul ia valori între −1 și 1. Nici această formă nu este suficientă dacă ținem cont de faptul că, pe ecran, pixelul aflat în colțul din stânga sus are coordonatele (0,0) și nu cel din stânga jos cum ar fi fost normal. Ținând cont de această ultimă observație, y se calculează astfel:

y: = getmaxy/2 − sin(−pi + 2 * pi/getmaxx * i) * getmaxy/2

Când sinusul ia valoarea −1, y va fi getmaxy, iar când sinusul ia valoarea 1, y va fi 0. Ținând cont de faptul că procedurile grafice solicită valorile x și y întregi, y se calculează prin rotunjirea valorii obținute (funcția ROUND). Trasarea graficului se face unind printr-un segment orice pereche de puncte consecutive care aparțin graficului funcției. Mecanismul de trasare a acestui grafic rezultă și din Figura 10.3.

Observație. Nu avem aceeași unitate de măsură pe OX și OY și nici nu ar fi posibil acest fapt. Din acest motiv, graficele pot apărea uneori deformate.

Cum putem trasa graficul unei funcții oarecare?

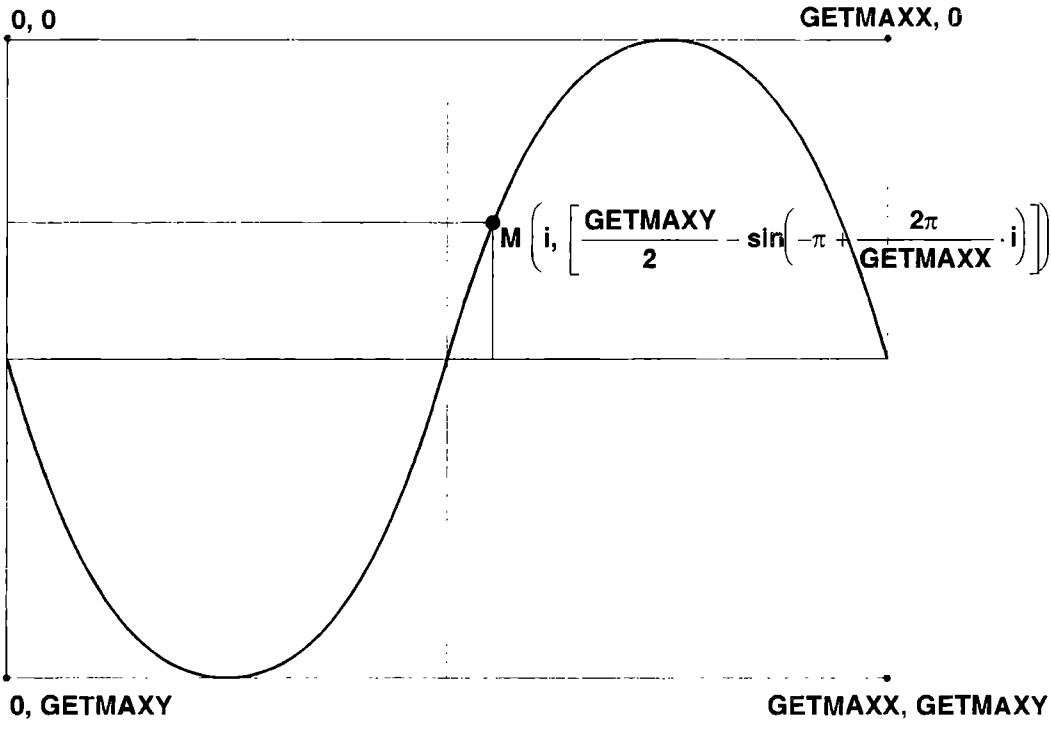


Figura 10.3.

Programul care urmează trasează graficul unei funcții care se găsește în unitatea de program FSTAND și care este definită pe intervalul [a,b]. Programul trebuie să funcționeze oricare ar fi funcția conținută în unitatea de program.

Cum se procedează? Inițial, se citește domeniul de definiție al funcției (valorile a și b), se trece în mod grafic și se trasează axa OX. Trasarea axei OY se face numai în situația în care intervalul [a,b] conține valoarea 0 (în caz contrar, axa OY nu este vizibilă pe ecran). Așa cum de altfel a rezultat și din programul anterior, graficul se trasează utilizând getmaxx + 1 puncte (de la 0 la getmaxx). Intervalul [a,b] se împarte în getmaxx părți egale (avem pasul de lucru (b−a)/getmaxx). Se calculează f(a + pas∗i), pentru i = 0, 1,...getmaxx. Când i ia valoarea 0 se calculează f(a) iar când i ia valoarea getmaxx se calculează f(a + (b−a)/getmaxx∗getmaxx) = f(a + b−a) = f(b). Toate aceste valori se reține într-un vector cu componente numere reale, numit VALF. Problema care apare este de a stabili unitatea de măsură pe axa OY. Problema nu este banală. Pentru programul anterior se știa faptul că funcția sinus ia valori între −1 și 1, dar aici avem o funcție oarecare, pentru care nu cunoaștem în prealabil mulțimea valorilor ei. Mai mult decât atât, chiar dacă am cunoaște funcția, dacă aceasta are o expresie complicată nu putem, prin procedee strict matematice, să determinăm mulțimea valorilor ei. Din cele prezentate, rezultă faptul că singurul mecanism posibil de lucru este de a calcula în vectorul VALF toate valorile pe care le ia funcția în punctele din intervalul considerat, după care

urmează stabilirea unității de măsură pe axa OY. Pentru aceasta, trebuie să vedem care este valoarea minimă și care este valoarea maximă luată de funcție (în program acestea se calculează în cadrul variabilelor MINF și MAXF).

Din păcate, nu în toate cazurile acestea sunt simetrice față de axa OX (cum se întâmplă în cazul funcției sinus pe intervalul considerat, unde valoarea minimă este −1 și valoarea maximă este 1). Aici, este posibil să avem valoarea maximă 100 și cea minimă −10. Pentru a putea reprezenta corect aceste valori, convenim următoarele: cea mai mare valoare în modul se reprezintă în partea de sus a ecranului (prin pixeli care au y = 0), dacă este pozitivă, sau în partea de jos a ecranului (prin pixeli care au y = getmaxy), în cazul când aceasta este negativă. Restul valorilor se reprezintă proporțional.

În concluzie, programul acționează astfel:

- se compară cele două valori (în modul);
  - în cazul în care cea mai mare valoare în modul este cea maximă, se înlocuiește fiecare valoare a funcției din VALF prin expresia: valf[i]:= getmaxy/2−getmaxy/(2*maxf)*valf[i], i = 0,1,...,getmaxx;
  - în cazul în care cea mai mare valoare în modul este cea negativă, se înlocuiește fiecare valoare a funcției din VALF prin expresia: valf[i]:= getmaxy/2−getmaxy/(2*abs(minf))*valf[i].

Ce am obținut astfel? În prima situație, valorii maxime a funcției îi corespunde:

valf[i] = getmaxy/2−getmaxy/(2*maxf)*maxf = 0,

iar în al doilea caz, valorii minime îi corespunde:

valf[i]: = getmaxy/2−getmaxy/(2*abs(minf))*min(f) = getmaxy,

dacă ținem cont de faptul că, în acest caz, minimul este negativ. În oricare din situații, restul valorilor se reprezintă proporțional pe ecran. În final, se trasează graficul prin unirea prin segmente a oricăror două perechi de puncte consecutive, secvență pe care, datorită simplității ei, nu o comentăm.

```
program curba;
uses crt,graph,utilg,fstand;
var a,b,pas,minf,maxf:real;
    valf:array[0..1000] of real;
    i:integer;

begin
  clrscr;
  writeln('domeniul de definitie [a,b] ');
  write('a=');readln(a);
  write('b=');readln(b);
  initg;
  moveto(0,getmaxy div 2);
  lineto(getmaxx,getmaxy div 2);
  if (a<0) and (b>0) then
  begin
    moveto(round(abs(a)/(abs(a)+b)*getmaxx),0);
    lineto(round(abs(a)/(abs(a)+b)*getmaxx),getmaxy)
  end;
  pas:=(b-a)/getmaxx;
  for i:=0 to getmaxx do valf[i]:=f(a+pas*i);
```

```
  minf:=valf[0]; maxf:=valf[0];
  for i:=1 to getmaxx do
  begin
    if minf>valf[i] then minf:=valf[i];
    if maxf<valf[i] then maxf:=valf[i];
  end;
  if abs(maxf)>=abs(minf)
  then for i:=0 to getmaxx do
         valf[i]:=getmaxy/2-getmaxy/(2*maxf)*valf[i]
  else for i:=0 to getmaxx do
         valf[i]:=getmaxy/2-getmaxy/(2*abs(minf))*valf[i]
  i:=0;
  moveto(0,round(valf[0]));
  repeat
    lineto(i,round(valf[i]));
    i:=i+1;
  until i>getmaxx;
  repeat
  until keypressed
end.
```

Așa cum am conceput până în acest moment programul, pentru a putea să-l rulăm trebuie să creăm în prealabil unitatea de program FSTAND care conține funcția ce se reprezintă grafic. De câte ori dorim să reprezentăm grafic o altă funcție, unitatea FSTAND trebuie recreată. Să recunoaștem că acest procedeu este destul de greoi și se pierde mult timp. Din păcate, limbajul TURBO PASCAL nu permite citirea unei funcții oarecare, căreia să-i putem calcula valorile. Ce artificii ar trebui să facem pentru ca acest fapt să devină posibil?

Există posibilitatea să scriem noi un program care să citească expresia unei funcții și să genereze o secvență de calcul care, la execuție, să calculeze valorile funcției în diverse puncte. Despre modul în care se realizează un astfel de program vom învăța în anul următor (sunt necesare cunoștințe care depășesc nivelul clasei a noua).

Pentru moment, vom folosi o altă metodă. În ce constă aceasta?

Programul GRAFIC_F se apelează de câte ori dorim să facem graficul unei noi funcții. Acesta realizează următoarele:

- apelează, spre a fi executat, programul CITF;
- apelează programul compilator TPC spre a compila programul de realizare a graficului unei funcții oarecare (program numit CURBA și care a fost prezentat anterior);
- apelează spre execuție programul CURBA.

```
program grafic_f;
uses dos;
{$m 4000,0,0}

begin
  swapvectors;
  exec('citf.exe','');
  swapvectors;
  exec('c:\tp\tpc.exe','curba');
```

```
    swapvectors;
    exec('curba.exe','');
    swapvectors;
  end.
```

Programul CITF creează unitatea de program FSTAND. Aceasta este creată ca un fișier text oarecare. Numele fișierului trebuie să aibă extensia PAS. Fiecare înregistrare a acestui fișier va conține o linie din unitatea de program. Astfel, se scriu pe rând liniile:

```
unit fstand;

interface
function f(x:real):real;

implementation
function f(x:real):real;
begin
```

În acest moment, programul citește valoarea funcției într-o variabilă de tip STRING. Se scrie în continuare:

```
f:=continutul variabilei STRING;
end;
end.
```

După închiderea fișierului FSTAND.PAS se apelează compilatorul TPC care compilează unitatea de program. În cazul în care la compilare apar erori, acestea provin de la introducerea incorectă a expresiei funcției. În acest caz, prin analiza codului prin care s-a terminat programul TPC (funcția DOSEXITMODE), se reia crearea fișierului FSTAND.PAS.

```
program citf;
{$m 4000,0,0}
uses dos;
var f:text;
    exp:string;
    tpc,numep:string;

begin
  assign(f,'fstand.pas');
  repeat
    writeln;
    rewrite(f);
    writeln(f,'unit fstand;');
    writeln(f,'interface');
    writeln (f,'function f(x:real):real;');
    writeln(f,'implementation');
    writeln (f,'function f(x:real):real;');
    writeln(f,'begin');
    write('f:=');readln(exp);
    write(f,'f:=');writeln(f,exp);
    writeln(f,'end;');
    writeln(f,'end.');
```

```
    close(f);
    tpc:=fsearch ('tpc.exe',getenv('path'));
    numep:='fstand.pas';
    swapvectors;
    exec(tpc,numep);
    swapvectors
  until dosexitcode=0
end.
```

În momentul în care avem unitatea FSTAND.PAS gata compilată se poate compila programul care realizează graficul. Care este motivul pentru care este necesară această compilare? În definitiv, am fi putut compila o singură dată programul CURBA. În realitate, programul obținut (în cod mașină) conține în cadrul lui toate subprogramele pe care le apelează. Din acest motiv, dacă nu compilăm programul CURBA în momentul în care avem la dispoziție noua unitate de program, nu realizăm ceea ce dorim.

În continuare prezentăm câteva din rezultatele obținute cu acest lanț de programe:

10.13.2. Desenarea curbelor plane

La matematică am învățat ecuația cercului cu centrul în origine, de rază dată. Considerăm reprezentarea cercului în plan. Este acesta un grafic de funcție? Evident nu, deoarece există valori ale lui x pentru care corespund două valori ale lui y (o paralelă la axa OY poate să taie cercul în două puncte). Atunci cum am putea proceda ca să trasăm un cerc? O primă posibilitate este să obținem reprezentarea grafică a cercului reprezentând grafic următoarele două funcții de mai jos:

$y^2=r^2-x^2$; $y_{1,2}=\pm(r^2-x^2)^{1/2}$; $f=(r^2-x^2)^{1/2}$, $x\in[-r,r]$;

$g=-(r^2-x^2)^{1/2}$, $x\in[-r,r]$

Nu este cam greu? Dar dacă o curbă are o ecuație mult mai complicată, din care nu mai rezultă așa de ușor funcțiile care trebuie reprezentate? Din acest motiv, s-a ajuns la concluzia că este necesar un alt mecanism prin care să fie dată o curbă. Astfel, putem alege (deloc ușor) două funcții, f și g, de variabilă t, cu t aparținând intervalului [a,b], astfel încât $x=f(t)$ și $y=g(t)$. Prin eliminarea lui t, se ajunge la ecuația inițială a curbei. Astfel, cercul poate fi reprezentat ca fiind dat de ecuațiile $x:=r*\sin(t)$ și $y:=r*\cos(t)$, cu t aparținând intervalului $[0,2*\pi]$. Eliminarea lui t se face ca mai jos, obținând astfel ecuația cercului.

$$\sin t=\frac{x}{r};\quad \cos t=\frac{y}{r};$$

$$\sin^2 t+\cos^2 t=1 \Rightarrow \frac{x^2}{r^2}+\frac{y^2}{r^2}=1 \Rightarrow x^2+y^2=r^2 \Rightarrow x^2+y^2-r^2=0$$

Problema care se pune în continuare este următoarea: se dă o curbă exprimată parametric, x=f(t) și y=g(t), t aparține lui [a,b], și se cere să se reprezinte grafic.

Citirea funcțiilor f și g se face ca în paragraful precedent, în care citeam o singură funcție. Programul GRAFIC_P gestionează întregul proces, după cum se vede mai jos:

```
program grafic_p;
uses dos;
{$m 4000,0,0}

begin
  swapvectors;
  exec('citfp.exe','');
  swapvectors;
  exec('c:\tp\tpc.exe','curbap');
  swapvectors;
  exec('curbap.exe','');
  swapvectors;
end.
```

Programul CITFP creează unitatea FSTAND (care conține expresiile funcțiilor f și g) și o compilează.

```
program citfp;
{$m 4000,0,0}
uses dos;
var f:text;
    exp:string;
    tpc,numep:string;

begin
  assign(f,'fstand.pas');
  repeat
    writeln;
    rewrite(f);
    writeln(f,'unit fstand;'); writeln(f,'interface');

    writeln (f,'function f(t:real):real;');
    writeln(f,'function g(t:real):real;');

    writeln(f,'implementation');
    writeln (f,'function f(t:real):real;');
    writeln(f,'begin');
    write('f:=');readln(exp);
    write(f,'f:=');writeln(f,exp);
    writeln(f,'end;');

    writeln (f,'function g(t:real):real;');
    writeln(f,'begin');
    write('g:=');readln(exp);
    write(f,'g:=');writeln(f,exp);
    writeln(f,'end;');

    writeln(f,'end.');
```

```
      close(f);
      tpc:=fsearch ('tpc.exe',getenv('path'));
      numep:='fstand.pas';
      swapvectors;
      exec(tpc,numep);
      swapvectors
    until dosexitcode=0
  end.
```

Programul CURBAP desenează curba pe monitor, după ce, în prealabil, cere domeniul de definiție al celor două funcții.

Care este mecanismul prin care se trasează desenul curbei?

În primul rând, intervalul [a,b] se împarte în o mie de părți egale. În fiecare punct astfel obținut, se calculează x=f(t) și y=g(t). În acest fel obținem coordonatele reale ale unui punct aparținând curbei, coordonate care se reține în cadrul a doi vectori XE (pentru valorile lui x) și YE (pentru valorile lui y). Coordonatele astfel obținute trebuie transformate în coordonate ecran. Pentru aceasta, se procedează astfel:

- se calculează valoarea minimă și maximă a lui x;
- se calculează valoarea minimă și maximă a lui y;
- se calculează r1 ca fiind variația lui x (din maximul lui x se scade valoarea minimă a lui x);
- se calculează, în mod asemănător, r2 ca fiind variația lui y;
- se calculează r ca fiind maximul dintre r1 și r2 împărțit la getmaxy.

Pentru ce am procedat astfel? Acest procedeu a fost aplicat pentru a stabili unitatea de măsură (astfel, dacă x a avut o variație mai mare, curba reprezentată va ocupa întreg spațiul disponibil pentru x, și numai cât este necesar pentru y, și invers, dacă y a avut o variație mai mare, curba va ocupa întreg spațiul disponibil pentru y și numai cât este necesar pentru x). Care este motivul pentru care r a fost obținut prin împărțirea la getmaxy și nu prin împărțirea la getmaxx ? Am dorit să obținem această reprezentare într-un pătrat și pătratul de latură maximă conținut pe ecran este cu getmaxy+1 puncte adresabile (dacă acceptam să reprezentăm curba pe dreptunghiul conținut de întregul ecran, un cerc ar fi arătat ca o elipsă...).

În continuare, se transformă efectiv coordonatele găsite în XE și YE în coordonate ecran. În acest scop, se folosesc formulele de mai jos:

```
xe[i]:=(xe[i]-minx)/r;
ye[i]:=getmaxy-(ye[i]-miny)/r;
end;
```

Să presupunem că x a avut variația maximă. În punctul în care x este maxim, xe[i] va fi:

(maxx-min)*getmaxy/(maxx-min)=getmaxy;

În punctul în care x este minim, xe[i] va fi 0. Restul valorilor se răspândesc proporțional în intervalul [0,getmaxy]. Întrucât variația lui y este mai mică decât aceea a lui x, ye[i] va avea valori în același interval, fără să-l acopere în întregime (în transformarea lui y se ține cont de faptul că un punct cu y minim

trebuie să se reprezinte în partea de jos a ecranului, deci va avea o valoare apropiată de getmaxy).

După această transformare, trasăm curba prin unirea punctelor consecutive prin segmente.

```
program curbap;
uses graph,utilg,fstand;
var nrt,i,j:integer;
    a,b,t,minx,miny,maxy,maxx,r1,r2,r:real;
    xe,ye:array[1..1000] of real;

begin
  write('a=');readln(a);
  write('b=');readln(b);
  nrt:=1000;
  t:=a;
  for i:=1 to nrt do
  begin
    xe[i]:=f(t);
    ye[i]:=g(t);
    t:=t+(b-a)/nrt;
  end;

  minx:=xe[1];maxx:=xe[1];
  miny:=ye[1];maxy:=ye[1];

  for i:=1 to nrt do
  begin
    if xe[i]<minx then minx:=xe[i];
    if xe[i]>maxx then maxx:=xe[i];
    if ye[i]<miny then miny:=ye[i];
    if ye[i]>maxy then maxy:=ye[i];
  end;

  initg;
  r1:=(maxx-minx);
  r2:=(maxy-miny);
  if r1>r2   then r:=r1/getmaxy
             else r:=r2/getmaxy;
  for i:=1 to nrt do
  begin
    xe[i]:=(xe[i]-minx)/r;
    ye[i]:=getmaxy-(ye[i]-miny)/r;
  end;
  moveto(round(xe[1]),round(ye[1]));
  for i:=2 to nrt do
  begin
    lineto(round(xe[i]),round(ye[i]));
  end;

  readln;
  closegraph;
end.
```

Un exemplu de reprezentare grafică se găsește în PLANȘA 5.

10.13.3. Rotații

În prezentul paragraf răspundem la întrebarea: cum rotim o figură plană?

Această problemă se reduce la una mai simplă, și anume: cum rotim un punct cu un unghi dat în jurul altui punct. Dacă știm să răspundem la această întrebare, putem roti o figură în jurul unui punct prin rotația punctelor care o caracterizează. De exemplu, pentru rotația unui pătrat în jurul unui punct, se rotesc punctele care constituie vârfurile acestuia și se trasează pătratul rotit prin intermediul acestor puncte. Fie figura 10.4., în care punctul de coordonate (x1,y1) se rotește în jurul punctului de coordonate (xc,yc) cu un unghi dat.

Dorim să obținem noile coordonate ale punctului P (x1,y1).

$$x = x_C + MT_1 = x_C + MP'\cos(\varphi+\theta) = x_C + MP'(\cos\varphi\ \cos\theta - \sin\varphi\ \sin\theta)$$

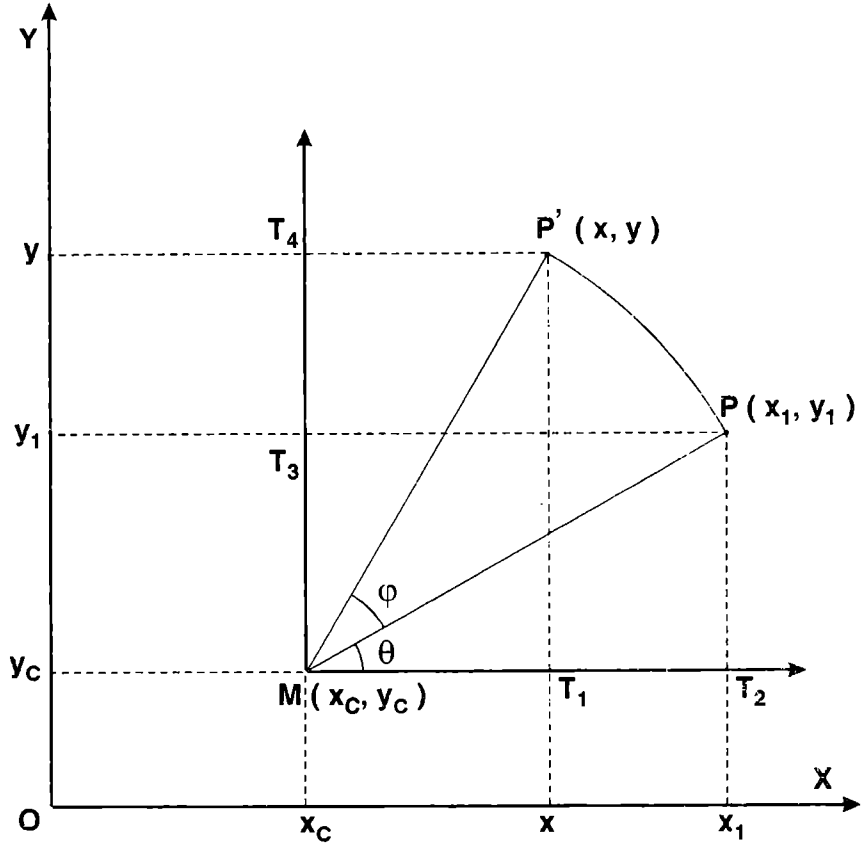


Figura 10.4.

Dar $MP = MP'$ (rază); $MT_2 = x_1 - x_C$; $MT_3 = y_1 - y_C$; $\cos\theta = MT_2/MP = (x_1 - x_C)/MP'$;

$\sin\theta = PT_2/MP = MT_3/MP = (y_1 - y_C)/MP' \Rightarrow$

$\Rightarrow x = x_C + MP'[\cos\varphi(x_1 - x_C)/MP' - \sin\varphi(y_1 - y_C)/MP'] \Rightarrow$

$\Rightarrow \mathbf{x = x_C + (x_1 - x_C)\cos\varphi - (y_1 - y_C)\sin\varphi}$

Analog $y = y_C + MT_4 = y_C + MP'\cos[90 - (\theta + \varphi)] = y_C + MP'\sin(\theta + \varphi) =$

$= y_C + MP'(\sin\theta\cos\varphi + \sin\varphi\cos\theta) = y_C + MP'[(y_1 - y_c)/MP']\cos\varphi +$

$+ [(x_1 - x_c)/MP']\sin\varphi) = y_C + (y_1 - y_C)\cos\varphi + (x_1 - x_C)\sin\varphi \Rightarrow$

$\Rightarrow \mathbf{y = y_C + (y_1 - y_C)\cos\varphi + (x_1 - x_C)\sin\varphi}$

Formulele găsite sunt conținute de procedura ROTPLAN care se găsește în cadrul unității UTILG.

```
procedure rotplan  (xc,yc,x1,y1:integer;
                    var x,y:integer;unghi:real);
begin
  x:=round(xc+(x1-xc)*cos(unghi)-(y1-yc)*sin(unghi));
  y:=round(yc+(x1-xc)*sin(unghi)+(y1-yc)*cos(unghi))
end;
```

Programul care urmează desenează un pătrat, îl plimbă pe orizontală și verticală, îl rotește în jurul unui punct al său. Pentru acestea, se folosește tehnica de animație care utilizează procedura setwritemode.

```
program trot;
uses graph,utilg,crt;
var x1,y1,x2,y2,x3,y3,x4,y4,x,y,l,i:integer;

procedure desen(x1,y1,x2,y2,x3,y3,x4,y4:integer);
begin
  moveto(x1,y1);
  lineto(x2,y2);
  lineto(x3,y3);
  lineto(x4,y4);
  lineto(x1,y1)
end;

begin
  initg;
  setwritemode(xorput);
  x1:=100; y1:=100;
  x2:=120; y2:=100;
  x3:=120; y3:=120;
  x4:=100; y4:=120;
  desen (x1,y1,x2,y2,x3,y3,x4,y4);
  for i:=1 to 50 do
  begin
    delay(100);
    desen(x1,y1,x2,y2,x3,y3,x4,y4);
    x1:=x1+1;
    x2:=x2+1;
    x3:=x3+1;
    x4:=x4+1;
    desen(x1,y1,x2,y2,x3,y3,x4,y4);
  end;
  desen(x1,y1,x2,y2,x3,y3,x4,y4);
  for i:=1 to 50 do
```

```
  begin
    y1:=y1+1;
    y2:=y2+1;
    y3:=y3+1;
    y4:=y4+1;
    desen(x1,y1,x2,y2,x3,y3,x4,y4);
    delay(100);
    desen(x1,y1,x2,y2,x3,y3,x4,y4);
  end;
  desen(x1,y1,x2,y2,x3,y3,x4,y4);
  x:=x1; y:=y1;
  l:=x2-x1;
  for i:=1 to 100 do
  begin
    desen(x1,y1,x2,y2,x3,y3,x4,y4);
    rotplan(x,y,x+l,y,x2,y2,pi*i/50);
    rotplan(x,y,x+l,y+l,x3,y3,pi*i/50);
    rotplan(x,y,x,y+l,x4,y4,pi*i/50);
    desen(x1,y1,x2,y2,x3,y3,x4,y4);
    delay(100);
  end;
  readln;
  closegraph;
end.
```

10.13.4. Curbe BSPLINE

Se consideră trei puncte A(x1,y1), B(x2,y2), C(x3,y3) (considerăm că punctele pot fi afișate pe ecran utilizând direct aceste coordonate). Fie curba dată prin ecuații parametrice:

$$x = f(t) = 0{,}5[(1-t)^2 x_1 + (1+2t-2t^2)x_2 + t^2 x_3]$$
$$y = g(t) = 0{,}5[(1-t)^2 y_1 + (1+2t-2t^2)y_2 + t^2 y_3]$$

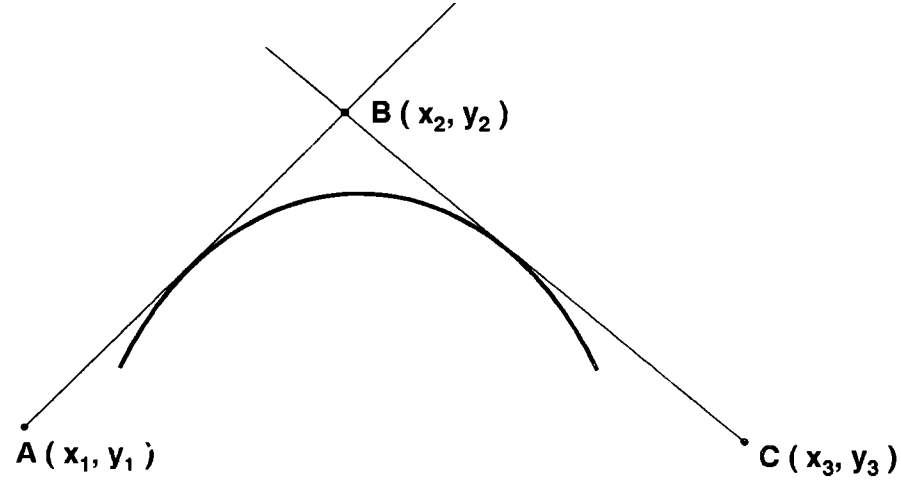


Figura 10.5.

O astfel de curbă se numește curba BSPLINE și are următoarele proprietăți:

- este tangentă segmentului AB în mijlocul său;
- este tangentă segmentului BC în mijlocul său;
- aproximează linia poligonală ABC, așa cum se vede în figura 10.5.

Punctele A, B, C se numesc puncte de ghidaj ale curbei.

Programul care urmează arată cum se poate desena pe monitor o astfel de curbă.

```
program bspline;
uses graph,utilg;

procedure xy(x1,y1,x2,y2,x3,y3:integer;t:real;var x,y:integer);
begin
  x:=round(0.5*(x1*(1-t)*(1-t)+x2*(1+2*t-2*t*t)+x3*t*t));
  y:=round(0.5*(y1*(1-t)*(1-t)+y2*(1+2*t-2*t*t)+y3*t*t));
end;

procedure trasez(x1,y1,x2,y2,x3,y3,nrp:integer);
var pas,t:real;
    x,y,i:integer;
begin
  t:=0;
  xy(x1,y1,x2,y2,x3,y3,t,x,y);
  moveto(x,y);
  pas:=1/nrp;
  for i:=1 to nrp do
  begin
    t:=t+pas;
    xy(x1,y1,x2,y2,x3,y3,t,x,y);
    lineto(x,y)
  end
end;

begin
  initg;
  moveto(0,0);
  lineto(100,100);
  lineto(200,0);
  trasez(0,0,100,100,200,0,100);
end.
```

Curbele BSPLINE sunt utilizate în realizarea desenelor. În principiu, se trasează desenul, format din linii frânte, pe o hârtie milimetrică. Atunci când se reprezintă pe monitor, se folosesc aceste curbe, ce permit realizarea unui desen cu forme rotunjite. Încercați să realizați astfel de desene spre a vă convinge.

10.14. Elemente de grafică 3D

În acest paragraf se prezintă câteva noțiuni strict necesare înțelegerii modului în care se pot reprezenta pe monitor obiecte din viața reală, și anume cele cu 3 dimensiuni. Multe din noțiunile prezentate sunt împrumutate din desenul

tehnic, datorită faptului că aceste obiecte se reprezentau pe hârtie cu mult timp înaintea apariției calculatoarelor.

Fie un punct M, de coordonate (x0, y0, z0), reprezentat în sistemul de axe OXYZ. Fie un plan P care are o poziție oarecare față de sistemul OXYZ. Atât punctul M cât și axele OX, OY, OZ se proiectează pe planul P. Proiecția unui punct oarecare pe planul P se face printr-o dreaptă paralelă cu o direcție dată (nu se realizează neapărat prin perpendiculara pe plan, deoarece nu este vorba de o proiecție ortogonală ci de una oblică).

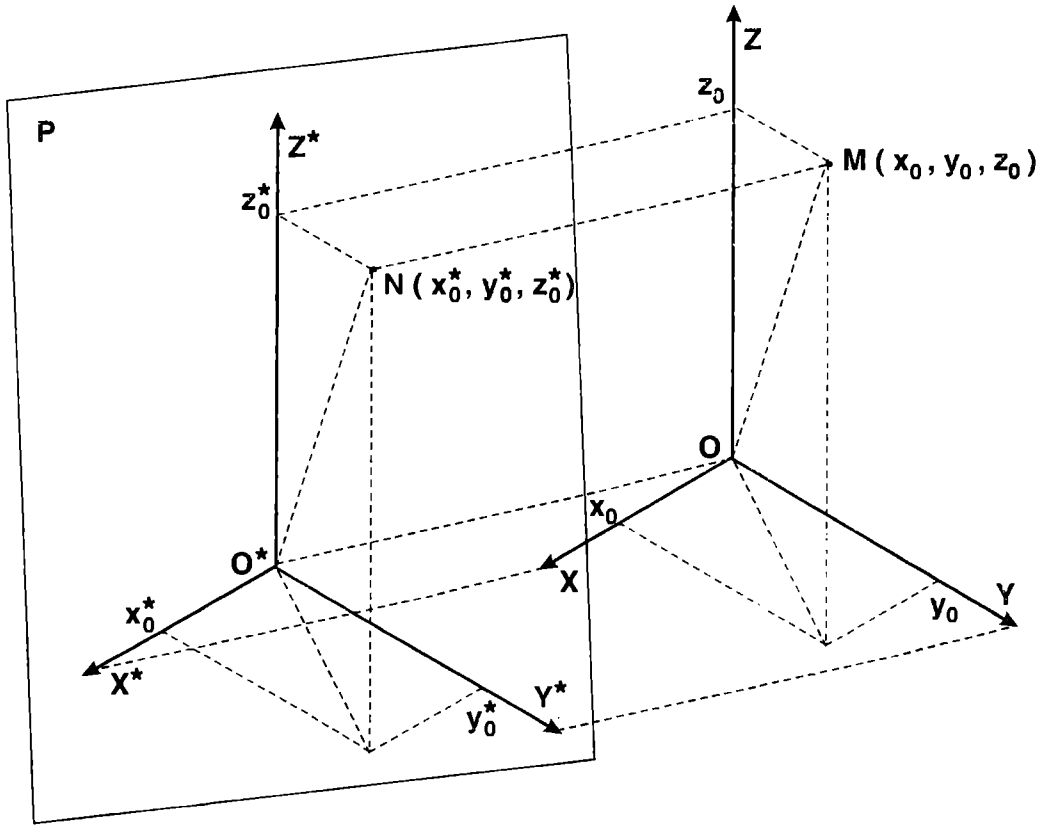


Figura 10.6.

Notăm cu N proiecția punctului M, cu O* proiecția punctului O, cu O*X*proiecția axei OX, cu O*Y* proiecția axei OY și cu O*Z* proiecția axei OZ. O*X*Y*Z* reprezintă un sistem de axe în planul P. Față de acest sistem de axe punctul N va avea coordonatele x0*, y0*, z0*. Evident, coordonatele punctului N nu coincid ca valoare cu coordonatele punctului M. Astfel, x0* va fi diferit de x0 ș.a.m.d.

Există procedee de exprimare a coordonatelor punctului N în funcție de coordonatele punctului M, însă acestea depășesc cu mult cunoștințele de

matematică din liceu și, după cum vom vedea în continuare, este chiar util să ne lipsim de acest calcul.

Din cele prezentate, reținem faptul că avem un sistem de axe (într-un plan oarecare P) $O^*X^*Y^*Z^*$, rezultat din proiecția sistemului de axe OXYZ. În acest sistem putem reprezenta un punct oarecare M(x,y,z) în sistemul OXYZ măsurând pur și simplu x, y, z unități pe axele O^*X^*, O^*Y^*, O^*Z^*. Așa cum putem reprezenta un punct în cadrul acestui sistem de axe, putem reprezenta și un obiect din spațiul cu 3 dimensiuni. În cele ce urmează vom lucra numai în planul P.

Problema care apare în continuare este următoarea: pentru a putea reprezenta un punct pe monitor, nu putem folosi în mod direct coordonatele punctului măsurate în sistemul $O^*X^*Y^*Z^*$, deoarece avem nevoie numai de 2 coordonate și nu de 3. În concluzie, trebuie găsită o modalitate de trecere de la 3 coordonate la 2 coordonate. Pentru a putea rezolva această problemă, se consideră un sistem de axe cu centrul în O^*, prin trasarea dreptelor perpendiculare O^*X0, O^*Y0 și se găsește o relație de transformare a coordonatelor punctului în acest sistem $((x_0^*, y_0^*, z_0^*) \rightarrow (x_0, y_0))$.

În figura 10.7. punctul N (x_0^*, y_0^*, z_0^*) în sistemul $O^*X^*Y^*Z^*$ are coordonatele (x_0, y_0) în sistemul $O^*X_0Y_0$. Notăm cu alfa, beta, gama unghiurile formate în sens trigonometric între axa O^*X_0 și axele O^*X^*, respectiv O^*Y^*, O^*Z^*. Trebuie găsite relațiile care dau pe x_0, y_0 în funcție de x_0^*, y_0^*, z_0^*, alfa, beta, gama.

Întrucât vom utiliza calculul vectorial, mai puțin folosit în orele de matematică, vom prezenta câteva noțiuni strict necesare înțelegerii demonstrației.

Fie M(xm,ym) și N(xn,yn) două puncte în sistemul de axe OXY. Acestor două puncte le corespund doi vectori $\vec{v_1}$ și $\vec{v_2}$ care se obțin prin unirea punctului O cu cele două puncte. După cum știm, cei doi vectori pot fi însumați utilizând regula paralelogramului. În acest mod, obținem vectorul $\vec{v}$ cu extremitățile în O și P. Se cere să aflăm coordonatele punctului P (figura 10.8.).

Avem următoarele relații: $xp = xm + xn$; $yp = ym + yn$.

Demonstrăm prima relație. Segmentele OM și NP sunt paralele și congruente (laturi opuse ale unui paralelogram). Rezultă că proiecțiile lor pe axa OX sunt congruente.

Avem deci relația: $xm = xp - xn$, de unde rezultă ce trebuia demonstrat. A doua relație se demonstrează în mod analog.

Revenim la problema noastră.

$O^*T_3T_1T_2$ este paralelogram. Rezultă

$$\overrightarrow{O^*T_1} = \overrightarrow{O^*T_3} + \overrightarrow{O^*T_2}.$$

$O^*T_1NT_4$ este paralelogram. Rezultă

$$\overrightarrow{O^*N} = \overrightarrow{O^*T_4} + \overrightarrow{O^*T_1} = \overrightarrow{O^*T_4} + \overrightarrow{O^*T_3} + \overrightarrow{O^*T_2}.$$

Utilizând ceea ce am demonstrat anterior putem transcrie relația scalar:

$x_0 = XT_3 + XT_2 + XT_4$; $y_0 = YT_3 + YT_2 + YT_4$,

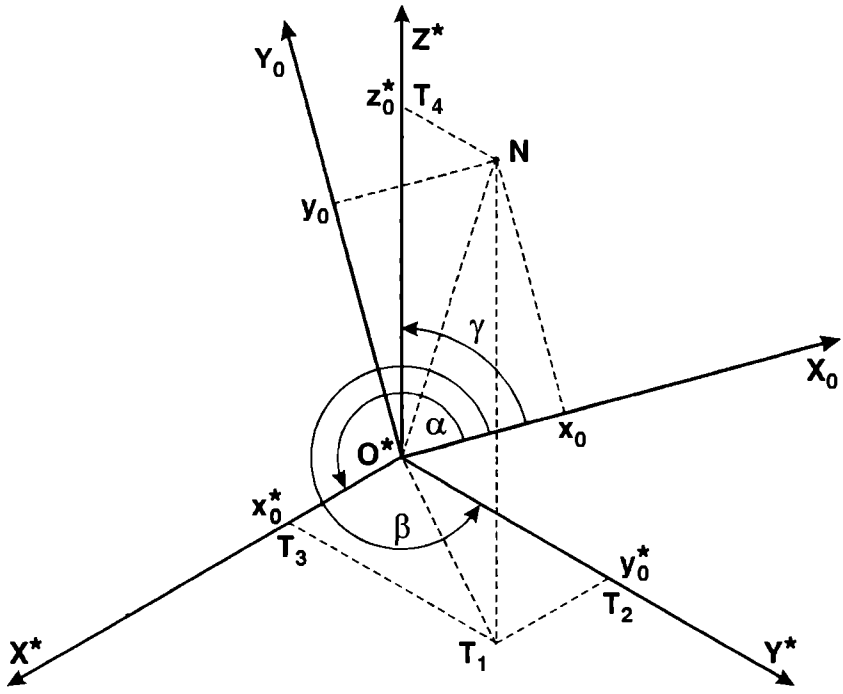


Figura 10.7.

unde XT_2, XT_3, XT_4 reprezintă coordonatele pe axa OX_0 ale punctelor T_2, T_3, T_4 iar YT_2, YT_3, YT_4 reprezintă coordonatele pe axa OY_0 ale punctelor T_2, T_3, T_4. Dar avem:

$XT_3 = x_0^* * \cos(alfa)$; $XT_2 = y_0^* * \cos(beta)$; $XT_4 = z_0^* * \cos(gama)$,

întrucât lungimea proiecției unui segment este egală cu lungimea segmentului înmulțită cu cosinusul unghiului format de dreapta care conține segmentul și dreapta pe care se face proiecția. În mod analog avem:

$YT_3 = x_0 * \cos(90-alfa) = x_0 * \sin(alfa)$; $YT_2 = y_0 * \cos(90-beta) = y_0 * \sin(beta)$;
$YT_4 = z_0 * \cos(90-gama) = z_0 * \sin(gama)$.

Am demonstrat astfel relațiile:

$x_0 = x_0^* * \cos(alfa) + y_0^* * \cos(beta) + z_0^* * \cos(gama)$;
$y_0 = x_0^* * \sin(alfa) + y_0^* * \sin(beta) + z_0^* * \sin(gama)$.

Aceste transformări le efectuează procedura d3d2g, care se găsește în unitatea de program UTILG.

```
procedure d3d2g(alfa,beta,gama:real;x1,y1,z1:real;var x,y:real);
begin
  x:=x1*cos(alfa)+y1*cos(beta)+z1*cos(gama);
  y:=x1*sin(alfa)+y1*sin(beta)+z1*sin(gama);
end;
```

Întrucât procedura are ca date de intrare unghiurile în radiani, a fost necesară construcția unei funcții, numită RAD, care are rolul de a transforma măsura unui unghi din grade în radiani.

```
function rad(x:integer):real;
begin
  rad:=pi*x/180
end;
```

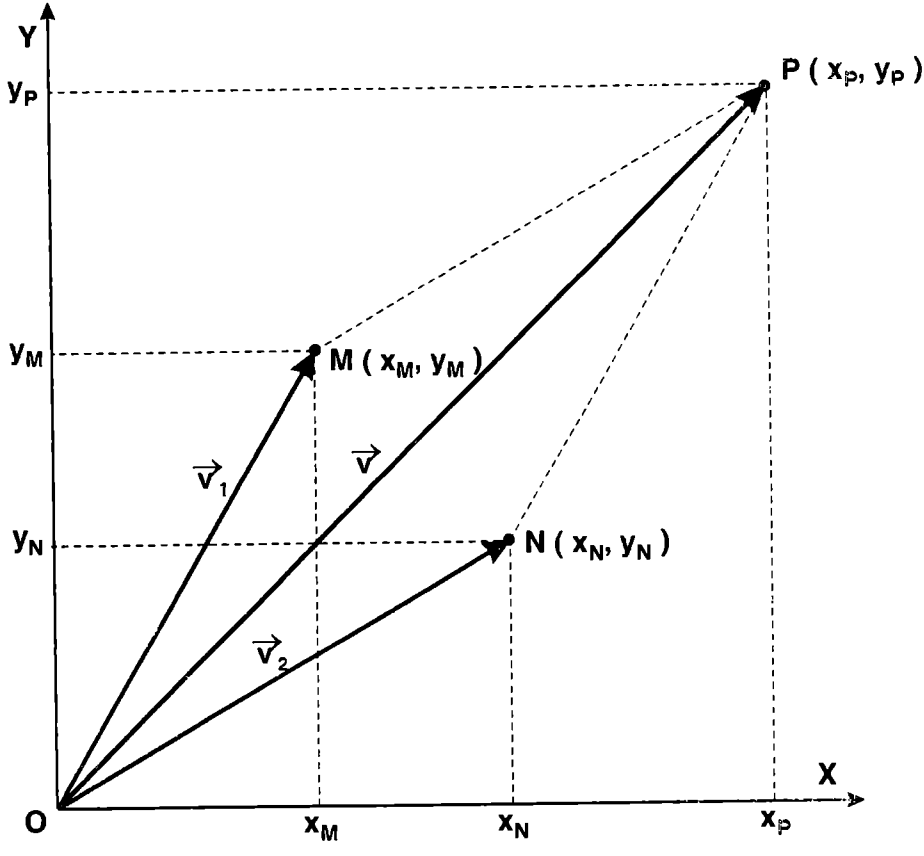


Figura 10.8.

În situația în care alfa = 0°, beta = 90°, gama = 45° avem proiecția **cabinet**, care dă imagini destul de apropiate de realitate.

În acest caz, formulele de calcul pentru x și y se simplifică:

x = x + z*cos(45); y = y + z*cos(45).

Pentru a face calcule mai rapid prezentăm și procedura d3d2, procedură de calcul al coordonatelor x și y în cazul proiecției cabinet. Este de menționat faptul că

această procedură furnizează direct coordonatele de afișare pe ecran (se ține cont de faptul că punctul de coordonate (0,0) se găsește în centrul ecranului).

```
procedure d3d2(x1,y1,z1:integer;var x,y:integer);
begin
  x:=getmaxx div 2 + round(x1+sqrt(2)*z1/4);
  y:=getmaxy div 2 - round(y1+sqrt(2)*z1/4)
end;
```

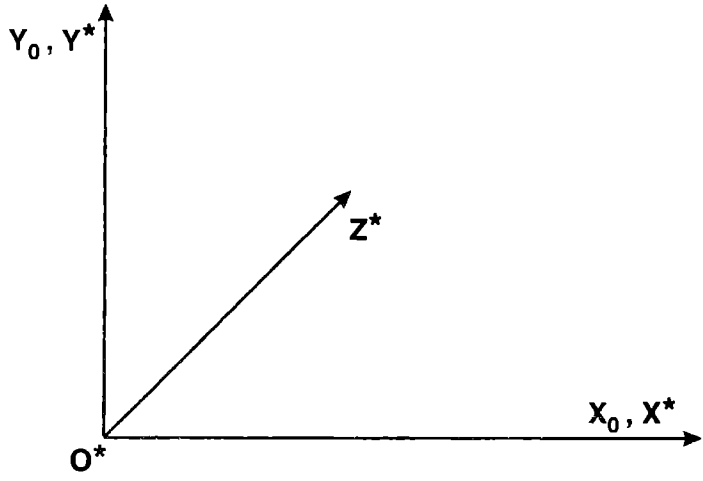


Figura 10.9.

Un alt triplet de valori pentru unghiurile alfa, beta și gama (în grade) este:

alfa = 165, beta = −135, gama = 135.

Acest triplet de valori este des folosit în reprezentarea suprafețelor în spațiu.

În continuare ne propunem să desenăm o prismă cu lungimile laturilor x1, y1, z1 unități. Modul de realizare a acestui program este foarte simplu. Se reprezintă prisma într-un sistem de axe și se trec pe desen coordonatele fiecărui punct.

Cele 3 coordonate ale fiecărui punct sunt convertite în 2 coordonate prin procedura D3D2G. Aceste puncte sunt unite prin segmente. În plus, programul cere și unghiurile alfa, beta gama, unghiuri necesare conversiei din D3 în D2. Încercați programul pentru tripletele:

alfa = 0, beta = 90, gama = 45 (proiecția cabinet);
alfa = 165, beta = −135, gama = 135.

Atenție! Programul nu testează dacă prin valorile x1,y1,z1 nu se face o tentativă de scriere în afara ecranului.

```
program prisma;
uses graph,utilg;
```

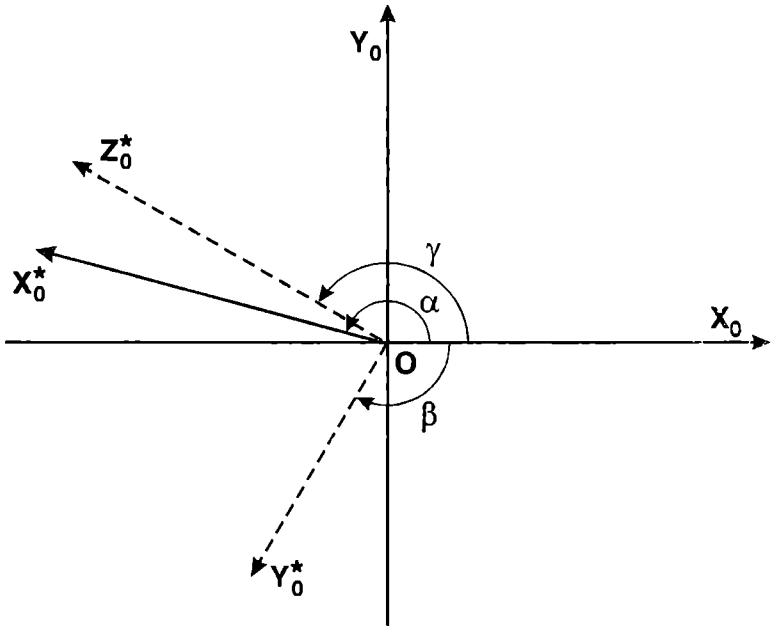


Figura 10.10.

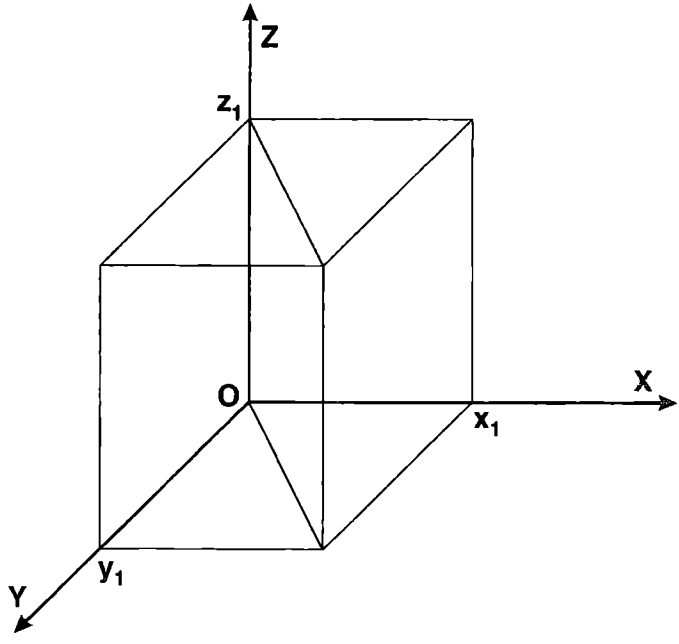


Figura 10.11.

```
var alfa,beta,gama:integer;
    x1,y1,z1,x,y:real;

begin
  write('alfa='); readln(alfa);
  write('beta='); readln(beta);
  write('gama='); readln(gama);
  write('x1='); readln(x1);
  write('y1='); readln(y1);
  write('z1='); readln(z1);
  initg;
  d3d2g(rad(alfa),rad(beta),rad(gama),0,0,0,x,y);
  moveto(getmaxx div 2 +round(x),getmaxy div 2+round(y));
  d3d2g(rad(alfa),rad(beta),rad(gama),0,y1,0,x,y);
  lineto(getmaxx div 2 +round(x),getmaxy div 2+round(y));
  d3d2g(rad(alfa),rad(beta),rad(gama),x1,y1,0,x,y);
  lineto(getmaxx div 2 +round(x),getmaxy div 2+round(y));
  d3d2g(rad(alfa),rad(beta),rad(gama),x1,0,0,x,y);
  lineto(getmaxx div 2 +round(x),getmaxy div 2+round(y));
  d3d2g(rad(alfa),rad(beta),rad(gama),0,0,0,x,y);
  lineto(getmaxx div 2 +round(x),getmaxy div 2+round(y));
  d3d2g(rad(alfa),rad(beta),rad(gama),0,0,z1,x,y);
  lineto(getmaxx div 2 +round(x),getmaxy div 2+round(y));
  d3d2g(rad(alfa),rad(beta),rad(gama),0,y1,z1,x,y);
  lineto(getmaxx div 2 +round(x),getmaxy div 2+round(y));
  d3d2g(rad(alfa),rad(beta),rad(gama),0,y1,0,x,y);
  lineto(getmaxx div 2 +round(x),getmaxy div 2+round(y));
  d3d2g(rad(alfa),rad(beta),rad(gama),x1,0,z1,x,y);
  moveto(getmaxx div 2 +round(x),getmaxy div 2+round(y));
  d3d2g(rad(alfa),rad(beta),rad(gama),x1,0,0,x,y);
  lineto(getmaxx div 2 +round(x),getmaxy div 2+round(y));
  d3d2g(rad(alfa),rad(beta),rad(gama),x1,0,z1,x,y);
  moveto(getmaxx div 2 +round(x),getmaxy div 2+round(y));
  d3d2g(rad(alfa),rad(beta),rad(gama),x1,y1,z1,x,y);
  lineto(getmaxx div 2 +round(x),getmaxy div 2+round(y));
  d3d2g(rad(alfa),rad(beta),rad(gama),0,y1,z1,x,y);
  lineto(getmaxx div 2 +round(x),getmaxy div 2+round(y));
  d3d2g(rad(alfa),rad(beta),rad(gama),x1,y1,z1,x,y);
  moveto(getmaxx div 2 +round(x),getmaxy div 2+round(y));
  d3d2g(rad(alfa),rad(beta),rad(gama),x1,y1,0,x,y);
  lineto(getmaxx div 2 +round(x),getmaxy div 2+round(y));
  d3d2g(rad(alfa),rad(beta),rad(gama),x1,0,z1,x,y);
  moveto(getmaxx div 2 +round(x),getmaxy div 2+round(y));
  d3d2g(rad(alfa),rad(beta),rad(gama),0,0,z1,x,y);
  lineto(getmaxx div 2 +round(x),getmaxy div 2+round(y));
  readln;
  closegraph
end.
```

10.4.1. Rotaţia unei figuri în spaţiu

În programul care urmează se roteşte un cub cu latura 50 în jurul axei OY. În principiu, se procedează astfel:

- se consideră 24 de poziţii intermediare ale cubului rotit în jurul axei OY cu unghiuri diferite;

- o poziție a cubului este desenată, ștearsă, după care se desenează poziția următoare, se șterge etc. (se folosește procedura setwritemode).

O poziție a cubului este reținută prin coordonatele celor 8 vârfuri ale sale. Fiind dată o poziție, se poate obține desenul cubului cu ajutorul procedurii DESEN. Pozițiile cubului sunt reținute în cadrul variabilei P. Pentru ca programul să meargă mai repede (dând astfel impresia de rotație), coordonatele vârfurilor cubului în cele 24 de poziții ale sale se calculează la început. Rotația cubului în jurul axei OY se reduce la o rotație în planul OXZ în jurul punctului O a pătratului care constituie baza cubului. Rotația pătratului se reduce la rotația punctelor care constituie vârfurile sale. Coordonatele vârfurilor feței opuse bazei se obțin din coordonatele vârfurilor bazei modificând coordonata y. Calculul coordonatelor vârfului unei poziții oarecare se face de către procedura COORD.

În concluzie se procedează astfel:
- se consideră o poziție particulară a cubului (în care 3 laturi ale sale coincid cu axele OX, OY, OZ) și se notează coordonatele fiecărui vârf al său;
- se rotește fiecare vârf al cubului cu un anumit unghi, obținând o poziție care se reține;
- procedăm în mod similar până se obțin toate cele 24 de poziții, care se memorează;
- fiecare poziție se desenează apoi se șterge, creându-se astfel impresia de rotație.

```
program g2;
uses graph,crt,utilg;
type figura=array[1..24,1..8,1..2] of integer;
var p:figura;
    i,j:integer; procedure coord(x,y,z:integer);

procedure coord(x,y,z:integer);
var i,j,xr,zr:integer;
begin
  j:=0; i:=1;
  repeat
    rotplan(0,0,0,0,xr,zr,pi*j/180);
    d3d2(xr,0,zr,p[i,1,1],p[i,1,2]);
    rotplan(0,0,x,0,xr,zr,pi*j/180);
    d3d2(xr,0,zr,p[i,2,1],p[i,2,2]);
    rotplan(0,0,x,z,xr,zr,pi*j/180);
    d3d2(xr,0,zr,p[i,3,1],p[i,3,2]);
    rotplan(0,0,0,z,xr,zr,pi*j/180);
    d3d2(xr,0,zr,p[i,4,1],p[i,4,2]);
    d3d2(0,y,0,p[i,5,1],p[i,5,2]);
    rotplan(0,0,x,0,xr,zr,pi*j/180);
    d3d2(xr,y,zr,p[i,6,1],p[i,6,2]);
    rotplan(0,0,x,z,xr,zr,pi*j/180);
    d3d2(xr,y,zr,p[i,7,1],p[i,7,2]);
    rotplan(0,0,0,z,xr,zr,pi*j/180);
    d3d2(xr,y,zr,p[i,8,1],p[i,8,2]) ;
    i:=i+1;
    j:=j+15;
```

```
  until i>24;
end;

procedure desen(i:integer);
var j:integer;
begin
  moveto(p[i,1,1],p[i,1,2]);
  for j:=2 to 4 do lineto(p[i,j,1],p[i,j,2]);
  lineto(p[i,1,1],p[i,1,2]);
  moveto(p[i,5,1],p[i,5,2]);
  for j:=6 to 8 do lineto(p[i,j,1],p[i,j,2]);
  lineto(p[i,5,1],p[i,5,2]);
  moveto(p[i,1,1],p[i,1,2]);lineto(p[i,5,1],p[i,5,2]);
  moveto(p[i,2,1],p[i,2,2]);lineto(p[i,6,1],p[i,6,2]);
  moveto(p[i,3,1],p[i,3,2]);lineto(p[i,7,1],p[i,7,2]);
  moveto(p[i,4,1],p[i,4,2]);lineto(p[i,8,1],p[i,8,2]);
end;

begin
  initg;
  coord(50,50,50);
  setwritemode(xorput);
  for j:=1 to 10 do
    for i:=1 to 24 do
    begin
      desen(i);
      delay(100);
      desen(i)
    end;
  desen(24);
  closegraph
end.
```

10.14.2. Reprezentarea unei suprafeţe în spaţiu

Se consideră o expresie de forma z=f(x,y), unde perechea (x,y) aparţine lui [a,b] * [c,d]. Funcţia f este o funcţie de două variabile.

Exemplu: z=x+y−3 (x, y) aparţine [0, 5]*[8, 10]. Pentru fiecare pereche (x, y), unde x aparţine intervalului [0, 5] şi y aparţine intervalului [8,10] se poate obţine o valoare pentru z. Astfel, x=2, y=9 rezultă z=2+9−3=8. Obţinem astfel un triplet (2, 9, 8), care reprezintă coordonatele unui punct în spaţiu. Avem o infinitate de triplete (care verifică această expresie), deci avem o infinitate de puncte date prin coordonatele lor. În cadrul cursurilor de matematică (din facultate) se arată că aceste puncte se găsesc plasate pe o suprafaţă din spaţiu (în cazul expresiei considerate, suprafaţa este un plan). Problema care se pune în continuare este următoarea: se dă o astfel de funcţie şi se cere să se reprezinte pe monitor suprafaţa pe care o reprezintă. Pentru asigurarea generalităţii programului, procedăm întocmai ca la desenarea graficelor de funcţii.

Un program numit GRAFIC_S lansează în execuţie un altul numit CITSUPR care are rolul de a crea şi compila o unitate de program numită FSTAND care conţine o funcţie dată de utilizator, după care compilează şi lansează în

execuție programul care desenează suprafața considerată. Rulați cele trei programe de mai jos și vă veți convinge.

```
program grafic_s;
uses dos;
{$m 4000,0,0}

begin
  swapvectors;
  exec('citsupr.exe','');
  swapvectors;
  exec('c:\tp\tpc.exe','supraf');
  swapvectors;
  exec('supraf.exe','');
  swapvectors;
end.

program citsupr;
{$m 4000,0,0}
uses dos;
var f:text;
    exp:string;
    tpc,numep:string;

begin
  assign(f,'fstand.pas');
  repeat
    writeln;
    rewrite(f);
    writeln(f,'unit fstand;');
    writeln(f,'interface');
    writeln (f,'function f(x,y:real):real;');
    writeln(f,'implementation');
    writeln (f,'function f(x,y:real):real;');
    writeln(f,'begin');
    write('f:='); readln(exp);
    write(f,'f:='); writeln(f,exp);
    writeln(f,'end;');
    writeln(f,'end.');
    close(f);
    tpc:=fsearch ('tpc.exe',getenv('path'));
    numep:='fstand.pas';
    swapvectors;
    exec(tpc,numep);
    swapvectors
  until dosexitcode=0
end.

program supraf;
uses graph,utilg,fstand;
var nrx,nry,alfa,beta,gama,i,j:integer;
    a,b,c,d,x,y,minx,miny,maxy,maxx,r1,r2:real;
    var xe,ye:array[1..50,1..50] of real;

begin
  write('numar valori x =');
```

```
  readln(nrx);
  write('numar valori y=');
  readln(nry);
  write('a='); readln(a);
  write('b='); readln(b);
  write('c='); readln(c);
  write('d='); readln(d);
  write('alfa='); readln(alfa);
  write('beta='); readln(beta);
  write('gama='); readln(gama);
  x:=a;
  for i:=1 to nrx do
  begin
    y:=c;
    for j:=1 to nry do
    begin
      d3d2g(rad(alfa),rad(beta),rad(gama),x,y,f(x,y),xe[i,j],
            ye[i,j]);
      y:=y+(d-c)/(nry-1)
    end ;
    x:=x+(b-a)/(nry-1)
  end;
  minx:=xe[1,1];
  maxx:=xe[1,1];
  miny:=ye[1,1];
  maxy:=ye[1,1];
  for i:=1 to nrx do
    for j:=1 to nry do
    begin
      if xe[i,j]<minx then minx:=xe[i,j];
      if xe[i,j]>maxx then maxx:=xe[i,j];
      if ye[i,j]<miny then miny:=ye[i,j];
      if ye[i,j]>maxy then maxy:=ye[i,j];
    end;
  initg;
  r1:=(maxx-minx)/ getmaxx;
  r2:=(maxy-miny)/getmaxy;
  for i:=1 to nrx do
    for j:=1 to nry do
    begin
      xe[i,j]:=(xe[i,j]-minx)/r1;
      ye[i,j]:=getmaxy-(ye[i,j]-miny)/r2;
    end;
  for i:=1 to nrx do
  begin
    moveto(round(xe[i,1]),round(ye[i,1]));
    for j:=2 to nry do lineto(round(xe[i,j]),round(ye[i,j]))
  end;
  for j:=1 to nry do
  begin
    moveto(round(xe[1,j]),round(ye[1,j]));
    for i:=2 to nrx do lineto(round(xe[i,j]),round(ye[i,j]))
  end;
  readln;
  closegraph;
end.
```

În continuare, arătăm modul în care am procedat pentru a desena suprafața asociată funcției. Considerăm sistemul de axe din spațiul cu 3 dimensiuni, în care este desenată o suprafață. Se alege o rețea de puncte, situate pe suprafața considerată, care vor fi reprezentate pe monitor, unite între ele prin segmente. Pentru alegerea rețelei, s împarte intervalul [a,b] în nrx părți egale și intervalul [c,d] în nry părți egale. Considerăm dreptunghiul [a,b]*[c,d] situat în planul XOY. Prin fiecare punct obținut în urma împărțirii intervalului [a,b] se duce o paralelă la axa OY și prin fiecare punct al intervalului [c,d] se duce o paralelă la axa OX. La intersecția celor două paralele se găsește un punct caracterizat de un anumit x, un anumit y. Pentru această pereche x,y se calculează z ca fiind f(x,y). În acest mod am obținut punctele care vor fi reprezentate. Toate punctele din rețea cu același x se unesc prin segmente, toate punctele din rețea cu același y se unesc prin segmente. Acestea se reprezintă pe monitor, obținându-se astfel imaginea rețelei. Urmăriți cele prezentate în Figura 10.12.

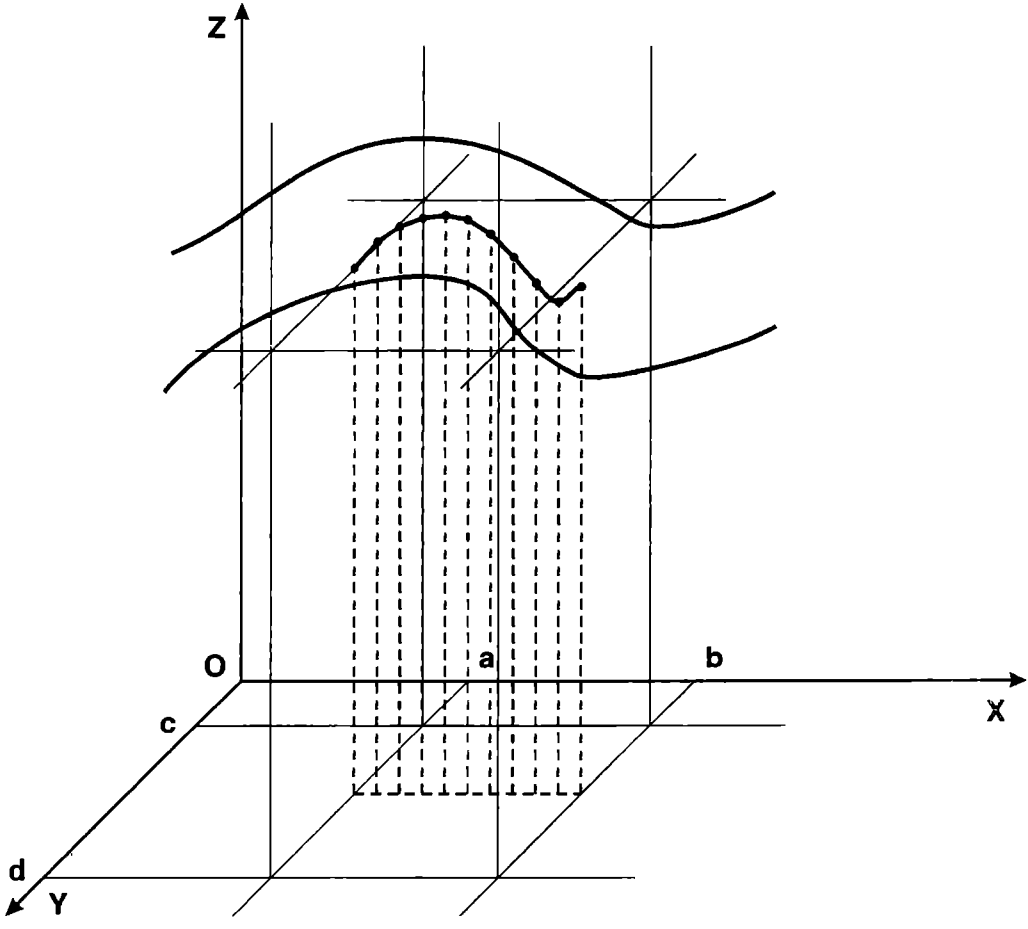


Figura 10.12.

A rămas de rezolvat o singură problemă, aceea de încadrare pe ecran a coordonatelor punctelor ce vor fi reprezentate. Din câte știm, pentru a putea reprezenta pe ecran un punct, avem nevoie numai de două coordonate și nu de trei. Pentru a se putea realiza această conversie, programul cere cele trei unghiuri între axe, alfa, beta, gama (unghiuri a căror semnificație a fost prezentată). Procedura D3D2G transformă cele trei coordonate ale fiecărui punct în două coordonate. Coordonata x a fiecărui punct este reținută în matricea xe, iar coordonata y a fiecărui punct este reținută în matricea ye. Fiecare coordonată este reținută într-o matrice deoarece avem nrx*nry puncte, deci xe[i,j] reprezintă coordonata x a punctului (i, j) cu i de la 1 la nrx și j de la 1 la nry. Problema se pune în mod similar pentru matricea ye. Acum, fiecare punct de pe suprafață este reprezentat prin două coordonate, dar cum le reprezentăm pe acestea pe ecran? Pentru fiecare element al matricei xe se calculează minimul și maximul, pe care le numim minx și miny. Calculăm două valori r1 = (maxx−minx)/getmaxx și r2 = (maxy−miny)/getmaxy;

Fiecărui element al matricei xe i se atribuie valoarea xe[i,j]: = (xe[i,j]−minx)/r1 și fiecărui element al matricei ye i se atribuie valoarea ye[i,j]: = getmaxy−−(ye[i,j]−miny)/r2.

Care este motivul pentru care am făcut aceste atribuiri?

Să presupunem un punct care are coordonata x minimă. Rezultă de aici că xe[i,j] va fi 0 (așa cum trebuie spre a fi reprezentat pe ecran). Dacă un punct va avea coordonata x maximă, atunci vom avea: xe[i,j]: = (maxx−minx)* *getmaxx/(maxx−minx) = getmaxx.

În mod similar avem și pentru y. Dacă un punct are coordonata y minimă vom avea ye[i,j] = getmaxy (se reprezintă în partea de sus a ecranului), iar dacă y este maxim vom avea ye[i,j] = 0. Un punct oarecare se reprezintă undeva pe ecran, proporțional cu valorile lui. Dacă avem coordonatele ecran, suprafața se poate reprezenta fără probleme.

Câteva exemple de suprafețe se găsesc în PLANȘELE 6, 7 și 8.

10.14.3. Coordonatele sferice și aplicațiile lor

Se consideră sistemul de axe din spațiul cu trei dimensiuni. Fie M (x,y,z). Se pune problema de a exprima coordonatele punctului M în funcție de alți trei parametri și anume:

- distanța de la origine la punctul M, notată R;
- unghiul format de proiecția dreptei OM cu axa OX, notat t;
- unghiul format de dreapta OM' cu OM, notat s;

În aceste condiții obținem:

OM' = OMcos(s) = Rcos(s); x = OM'cos(t) = Rcos(s)·cos(t);
y = OM'cos(90−t) = OM'sin(t) = Rcos(s)·sin(t); z = OMsin(s) = Rcos(s).

Deci: x = Rcos(s)·cos(t); y = Rcos(s)·sin(t); z = Rsin(s)

Acestea sunt coordonatele sferice.

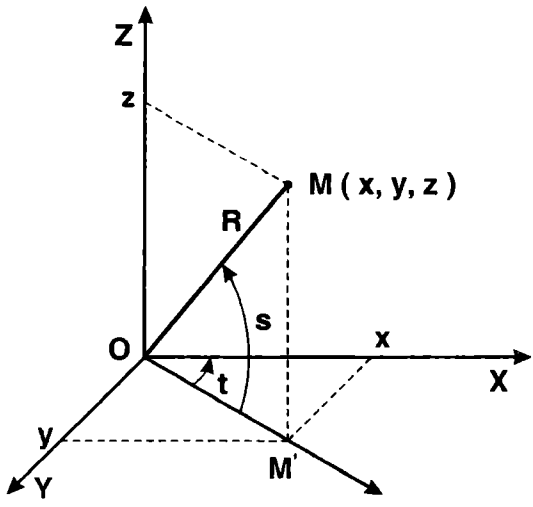


Figura 10.13.

Putem obține coordonatele oricărui punct M din spațiu, dacă $t \in [0, 2\pi)$ și $s \in [-\pi/2, \pi/2]$.

Fie planul OMZ. În acest plan, considerăm un sistem rectangular de axe OZ și OX_0 (coincide ca directie cu OM') și o curbă oarecare. După cum am văzut, o curbă poate fi exprimată și parametric, deci considerăm exprimarea sa parametrică, și anume: $x0 = x_0(u)$ și $z = z_0(u)$. Procedând la fel ca în calculul coordonatelor sferice, scriem coordonatele unui punct T de pe această curbă, în sistemul OXYZ. Ca atare vom avea:

$x = x_0(u) * \cos(t)$; $y = y_0(u) * \sin(t)$; $z = z_0(u)$.

Dacă rotim planul OMZ în jurul axei OZ, curba considerată în planul OMZ va descrie o suprafață de rotație. Rotația planului în jurul axei OZ se face prin modificarea unghiului t.

În general, o suprafață poate fi descrisă parametric sub forma:

$x = f(s,t)$; $y = g(s,t)$; $z = h(s,t)$.

În cele ce urmează, suntem interesați să vizualizăm o suprafață de rotație. Pentru a putea asigura generalitatea, vom folosi procedeul, de acum bine cunoscut, de a citi de la tastatură funcțiile f, g și h. Ținând cont de faptul că rotația planului se va face automat prin modificarea unghiului t, nu are sens să citim decât exprimarea parametrică a curbei în planul OMZ, scrierea completă a formulelor urmând să se facă în programul citsuprp (care creează unitatea de program FSTAND). Cum realizăm acest lucru se poate observa cu ușurință, analizând programul sursă. Modul de desenare a suprafeței de rotație este același cu cel folosit în paragraful anterior pentru a desena o suprafață dată neparametric.

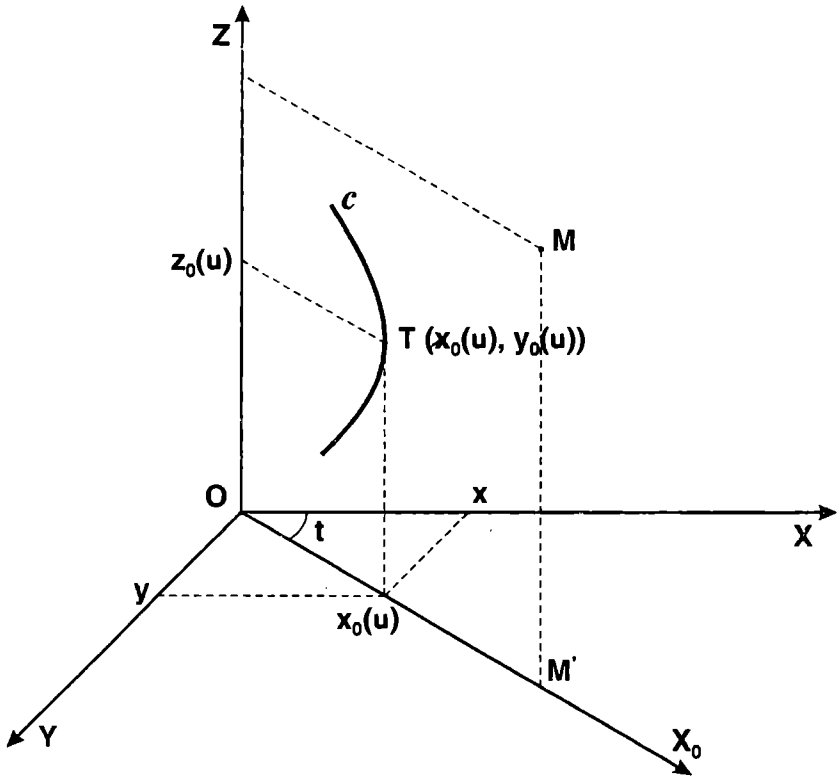


Figura 10.14.

```
program grafic_sr;
uses dos;
{$m 4000,0,0}

begin
  swapvectors;
  exec('citsuprp.exe','');
  swapvectors;
  exec('c:\tp\tpc.exe','suprafp');
  swapvectors;
  exec('suprafp.exe','');
  swapvectors;
end.

program citsupr;
{$m 4000,0,0}
uses dos;
var f:text;
    exp:string;
    tpc,numep:string;

begin
  assign(f,'fstand.pas');
```

```
  repeat
    writeln;
    rewrite(f);
    writeln(f,'unit fstand;');
    writeln(f,'interface');
    writeln (f,'function f(x,y:real):real;');
    writeln(f,'implementation');
    writeln (f,'function f(x,y:real):real;');
    writeln(f,'begin');
    write('f:='); readln(exp);
    write(f,'f:='); writeln(f,exp);
    writeln(f,'end;');
    writeln(f,'end.');
    close(f);
    tpc:=fsearch ('tpc.exe',getenv('path'));
    numep:='fstand.pas';
    swapvectors;
    exec(tpc,numep);
    swapvectors
  until dosexitcode=0
end.

program suprafp;
uses graph,utilg,fstand;
var nrs,nrt,i,j:integer;
    a,b,c,d,s,t,mins,mint,maxs,maxt,r1,r2,
    alfa,beta,gama:real;
    var xe,ye:array[1..50,1..50] of real;

begin
  write('numar valori s ='); readln(nrs);
  write('numar valori t='); readln(nrt);
  write ('a='); readln(a);
  a:=rad(trunc(a));
  write('b=');readln(b);
  b:=rad(trunc(b)); c:=0; d:=2*pi;
  alfa:=rad(210); beta:=rad(-30);gama:=rad(90);
  s:=a;
  for i:=1 to nrs do
  begin
    t:=c;
    for j:=1 to nrt do
    begin
      d3d2g(alfa,beta,gama,f(s,t),g(s,t),h(s,t),xe[i,j],
            ye[i,j]);
      t:=t+(d-c)/(nrt-1)
    end ;
    s:=s+(b-a)/(nrs-1)
  end;
  mins:=xe[1,1]; maxs:=xe[1,1];
  mint:=ye[1,1]; maxt:=ye[1,1];
  for i:=1 to nrs do
    for j:=1 to nrt do
    begin
      if xe[i,j]<mins then mins:=xe[i,j];
      if xe[i,j]>maxs then maxs:=xe[i,j];
```

```
      if ye[i,j]<mint then mint:=ye[i,j];
      if ye[i,j]>maxt then maxt:=ye[i,j];
    end;
  initg;
  r1:=(maxs-mins)/ getmaxx;
  r2:=(maxt-mint)/getmaxy;
  for i:=1 to nrs do
    for j:=1 to nrt do
    begin
      xe[i,j]:=(xe[i,j]-mins)/r1;
      ye[i,j]:=getmaxy-(ye[i,j]-mint)/r2;
    end;
  for i:=1 to nrs do
  begin
    moveto(round(xe[i,1]),round(ye[i,1]));
    for j:=2 to nrt do lineto(round(xe[i,j]),round(ye[i,j]))
  end;
  for j:=1 to nrt do
  begin
    moveto(round(xe[1,j]),round(ye[1,j]));
    for i:=2 to nrs do lineto(round(xe[i,j]),round(ye[i,j]))
  end;
  readln;
  closegraph;
end.
```

În PLANȘELE 9, 10 și 11 puteți observa câteva rezultate.

10.14.4. Prezentarea integrală a unității UTILG

```
unit utilg;

interface
uses graph;
var gdriver,gmode:integer;

procedure initg;
function rad(x:integer):real;

procedure rotplan(xc,yc,x1,y1:integer; var x,y:integer;
                  unghi:real);
procedure d3d2(x1,y1,z1:integer; var x,y:integer);
procedure d3d2n(x1,y1,z1:integer; var x,y:integer);

procedure citimg(x,y:integer;fdim,fimg:string);
procedure scrimg(x1,y1,x2,y2:integer; fdim,fimg:string;
                  var cod:byte);
procedure d3d2g(alfa,beta,gama:real; x1,y1,z1:real;
                  var x,y:real);

implementation
procedure initg;
begin
```

```
  gdriver:=detect;
  initgraph(gdriver,gmode,'c:\tp\bgi');
  if graphresult<>0 then
  begin
    writeln('tentativa esuata');
    halt
  end
end;

function rad;
begin
  rad:=pi*x/180
end;

procedure d3d2;
begin
  x:=getmaxx div 2 +round(x1+sqrt(2)*z1/4);
  y:=getmaxy div 2 -round(y1+sqrt(2)*z1/4)
end;

procedure d3d2n;
begin
  x:=round(x1+sqrt(2)*z1/4);
  y:=round(y1+sqrt(2)*z1/4)
end;

procedure rotplan(xc,yc,x1,y1:integer; var x,y:integer;
                  unghi:real);
begin
  x:=round(xc+(x1-xc)*cos(unghi)-(y1-yc)*sin(unghi));
  y:=round(yc+(x1-xc)*sin(unghi)+(y1-yc)*cos(unghi))
end;

procedure citimg;
var f1:text;
    f2:file;
    dim:integer;
    a:pointer;
begin
  assign(f1,fdim);
  assign(f2,fimg);
  reset(f1);
  read(f1,dim);
  reset(f2,dim);
  getmem(a,dim);
  blockread(f2,a^,1);
  putimage(x,y,a^,xorput);
  close(f1);
  close(f2);
end;

procedure scrimg;
var a:pointer;
    dim:integer;
    f1:text;
    f2:file;
```

```
begin
  assign(f1,fdim);
  assign(f2,fimg);
  dim:=imagesize(x1,y1,x2,y2);
  if dim<>0 then
  begin
    getmem(a,dim);
    getimage(x1,y1,x2,y2,a^);
    rewrite(f1);
    rewrite(f2,dim);
    write(f1,dim);
    blockwrite(f2,a^,1);
    close(f1);
    close(f2);
    cod:=0
  end
  else cod:=1
end;

procedure d3d2g;
begin
  x:=x1*cos(alfa)+y1*cos(beta)+z1*cos(gama);
  y:=x1*sin(alfa)+y1*sin(beta)+z1*sin(gama);
end;
```

10.15. Probleme propuse

1. Desenați 16 benzi colorate cu nuanțe diferite de roșu.

2. Fie secvența: setbkcolor(white); setcolor(black); moveto(0,0); lineto(50,0).
Care vor fi culorile de fond și de culoare pentru segmentul considerat?

3. Trasați toate tipurile de linii cunoscute.

4. Scrieți un program care să execute un desen la alegerea dumneavoastră.

5. Scrieți o procedură care deplasează un punct curent în 8 direcții. Deplasarea punctului într-o direcție anume se va face prin apăsarea unei taste prestabilite. În cazul apăsării continue a tastei, punctul va avea o deplasare continuă. Procedura va primi ca parametru de intrare o variabilă booleană care va indica dacă punctul lasă urmă sau nu. Atenție la ieșirea din ecran!

6. Utilizând eventual procedura de la punctul 5, realizați un program care ne permite să desenăm prin apăsarea tastelor (ștergerea unei figuri se va face prin a plimba punctul, care nu lasă urmă, asupra ei). Programul va fi înzestrat și cu opțiuni de stabilire a culorilor de fond și de desen.

7. Prin execuția programului de la problema 6 realizați un desen „artistic".

8. Salvați într-un fișier imaginea creată de dumneavoastră. Verificați salvarea, prin readucerea imaginii pe ecran.

9. Se consideră un fișier de personal în care fiecare înregistrare conține

numele unei persoane și vârsta sa. Realizați o situație statistică privind repartizarea persoanelor pe grupe de vârstă (0-20, 21-30, 31-40 ș.a.m.d.). Situația se va realiza în două feluri:

- prin utilizarea procedurii BAR3D;
- prin utilizarea procedurii PIESLICE.

10. Realizați un meniu prin utilizarea unei ferestre grafice. Selectarea opțiunii se va face prin plimbarea unei bare luminoase deasupra opțiunilor.

11. Aceeași problemă ca la problema 10, dar selectarea opțiunii se va face prin plasarea unui cadru dreptunghiular deasupra opțiunii, urmată de apăsarea tastei ENTER.

12. Realizați un program în care o bilă se mișcă aleator pe ecran. Programul va fi conceput de trei ori, utilizând cele trei variante de animație.

13. Realizați un program în care un omuleț stilizat își mișcă picioarele și mâinile.

14. Afișați în mod grafic numele persoanelor conținute în fișierul PERSONAL (vezi problemele de la fișiere).

15. Dintr-un pachet de cărți de joc se extrag aleator 4, care vor fi afișate pe ecran.

16. Simulați un joc de cărți în care un participant la joc este calculatorul.

17. Exprimați grafic noțiunea de arc capabil de un unghi dat.

18. Rezolvați grafic problema mișcării în plan a unui punct, astfel încât suma pătratelor distanțelor la două puncte fixe este o constantă.

19. Considerați, la alegere, o problemă de loc geometric din manual. Întocmiți un material didactic pentru orele de matematică, în care să se vadă efectiv mișcarea pe ecran.

20. Realizați prin programe proprii graficul funcției de gradul doi. Se va considera restricția [a, b] a domeniului de definiție.

21. Rotiți un segment în jurul unuia din capetele sale, cu un unghi alfa (la realizarea desenului țineți cont de raportul aspect).

22. Rotiți un triunghi echilateral în jurul ortocentrului său. Efectuați mai multe rotații cu un unghi convenabil ales, până când se ajunge la figura inițială. În cazul în care figura rotită nu este ștearsă, cum arată figura astfel creată?

23. Reprezentați în spațiu o piramidă patrulateră regulată.

24. Rotiți piramida de la punctul anterior în jurul înălțimii.

25. Rotiți, pe rând, un cub în jurul celor trei axe.

26. Desenați un con și un trunchi de con.

27. Reprezentați o secțiune plană (nu neapărat paralelă cu baza) într-o piramidă.

28. Rotiți desenul din problema anterioară în jurul înălțimii.

ANEXA

Utilizarea meniurilor în limbajul TURBO PASCAL

Limbajul TURBO PASCAL este înzestrat cu un sistem de meniuri care permite lucrul cu textele sursă, depanarea programelor, informări asupra diverselor instrucțiuni și proceduri pe care le conține mediul integrat (editor de texte, compilator, depanator, sistem de meniuri etc.). Există mai multe forme de a prezenta toate acestea. În ce ne privește, vom prefera prezentarea acestuia în funcție de operațiile cele mai frecvente care se efectuează cu ajutorul meniurilor.

Fereastra de lucru

Orice text sursă PASCAL se scrie într-o fereastră. De cele mai multe ori se lucrează cu una singură și acesta este cazul pe care-l vom discuta în continuare. De regulă (dacă nu au fost precizate alte opțiuni) la apelul editorului și compilatorului TURBO PASCAL se deschide o fereastră care are un nume pus de editor și anume NONAMExx.pas (xx reprezintă un număr de ordine). Există și posibilitatea de a deschide direct o fereastră cu un anumit nume (apelul se face cu TURBO nume).

În această fereastră se scrie textul sursă. Pentru început reținem faptul că la comanda CTRL + F9 textul este compilat și rulat. Pentru a vedea rezultatele rulării se tastează ALT + F5 iar pentru a reveni la textul sursă se tastează ENTER sau ALT + F5.

În partea de sus a ferestrei apare meniul cu care este înzestrat mediul integrat al limbajului. Astfel apar opțiunile următoare:

```
FILE EDIT SEARCH RUN COMPILE DEBUG OPTIONS WINDOW HELP
```

Avem două posibilități de a selecta o opțiune:

- se tastează F10, apoi ne deplasăm cu ajutorul săgeților pe una din opțiunile dorite;
- se tastează ALT + litera de început a opțiunii (de exemplu, pentru opțiunea FILE tastăm ALT + F);
- în ambele cazuri, după poziționarea pe o anumită opțiune se tastează ENTER iar pentru revenire se tastează ESC.

Cum lucrăm eficient utilizând sistemul de meniuri ?
Întrucât în timpul lucrului pot apărea o serie de surprize neplăcute (blocarea

calculatorului, întreruperea curentului, etc.) este bine să salvăm textul la fiecare câteva rânduri scrise. Cum procedăm pentru aceasta?

1. Se apelează meniul FILE și se selectează opțiunea SAVE (după care se tastează ENTER).
2. Se tastează direct F2 (aceasta este una din tastele „fierbinți", în sensul că este foarte folosită).

În cazul în care fereastra are un nume implicit, se deschide automat o fereastră în care suntem întrebați sub ce nume să se facă salvarea. Îndată ce tastăm un nume se face salvarea iar fereastra în care lucrăm capătă numele sub care a fost salvată.

Dar să analizăm și alte opțiuni pe care le avem la dispoziție în cadrul meniului FILE.

Meniul FILE

OPEN	F3
NEW	
SAVE	F2
SAVE AS...	
SAVE ALL	
CHANGE DIR	
PRINT	
GET INFO	
DOS SHELL	
EXIT	ALT+X

Opțiunea OPEN permite deschiderea unei alte ferestre. Dacă am selectat această opțiune, se deschide o fereastră în care suntem întrebați care este numele ferestrei care să se deschidă. După ce am dat numele, se deschide acea fereastră, în care se încarcă un text sursă (dacă pe suport se găsește un fișier sursă cu acel nume și extensia PAS) sau, în caz contrar, se deschide fereastra cu numele solicitat, în fereastră nefiind afișat nici un text. Deschiderea unei ferestre se poate face și prin apăsarea tastei „fierbinți" F3. Dacă se dorește deschiderea unei ferestre fără nici un text, cu numele implicit NONAME, aceasta se poate face prin utilizarea opțiunii NEW.

În mod normal, salvarea textului sursă se face în catalogul curent (de unde s-a făcut apelul TURBO). Dacă se dorește salvarea textului în alt catalog (subcatalog) se apelează opțiunea SAVE AS. La apelul acestei opțiuni se deschide o fereastră de dialog în care se precizează și calea pe care se va găsi fișierul salvat. Tot această opțiune se face și în situația în care se dorește salvare în paralel cu schimbarea numelui.

Opțiunea SAVE ALL permite salvarea conținutului tuturor ferestrelor cu care se lucrează la un moment dat. Lucrul cu mai multe ferestre va fi studiat într-un alt paragraf, motiv pentru care ne oprim aici cu explicațiile.

Opțiunea CHANGE DIR permite deplasarea într-un subcatalog al catalogului

curent. La apelul acestei opțiuni apare o fereastră în care se vizualizează subcataloagele. Dacă se tastează TAB ne putem deplasa în această arborescență.

Opțiunea PRINT permite tipărirea textului sursă la imprimantă (la apelul acestei opțiuni imprimanta trebuie să fie deschisă, altfel apare un mesaj de eroare care se anulează prin apăsarea tastei ENTER).

Opțiunea DOS SHELL permite ieșirea rapidă în sistemul DOS. Aici se pot executa diverse comenzi DOS, după care se revine prin tastarea cuvântului EXIT. Conținutul ferestrei (ferestrelor) în care am lucrat rămâne nemodificat.

Dacă se dorește ieșirea definitivă din mediul integrat TURBO PASCAL se tastează ALT + X sau se apelează opțiunea EXIT a meniului FILE.

Facilități de editare

Meniul SEARCH

Acesta are următoarele opțiuni care ușurează procesul de editare:

FIND
REPLACE
SEARCH AGAIN
GO TO LINE NUMBER
FIND PROCEDURE
FIND ERROR

Opțiunea FIND

Are rolul de a căuta un șir în textul programului sursă. În mod practic, se afișează o fereastră în care suntem solicitați să introducem textul care se va căuta.

De asemenea, în fereastră se prezintă alte câteva opțiuni (se ajunge la ele prin apăsarea tastei TAB și se marchează cu spațiu):

- CASE SENZITIVE - dacă este marcată (cu X) se face distincție între literele mari și cele mici;
- WHOLE WORDS ONLY - consideră textul ca un cuvânt separat (prin blancuri sau alți separatori);
- REGULAR EXPRESION - textul poate conține și caractere de control;
- DIRECTION - stabilește direcția în care se face căutarea:
  - FORWARD - spre sfârșit;
  - BACKWARD - spre început;
- ORIGIN - stabilește locul de unde începe căutarea:
  - FROM CURSOR - începând de la cursor;
  - ENTIRE SCOPE - întregul domeniu.
- OK - s-au terminat opțiunile, se închide fereastra și se fac înlocuirile;
- CANCEL - se renunță la căutare;

- HELP - se dau informații despre căutare.

Opțiunea REPLACE

Înlocuiește un șir cu altul. În acest sens, se deschide o fereastră care permite să se scrie șirul care se înlocuiește și șirul care se va înlocui. În această fereastră se găsesc opțiunile care au fost prezentate anterior și altele care urmează:

- PROMPT ON REPLACE - la fiecare înlocuire se deschide o fereastră în care utilizatorul este întrebat dacă să se facă sau nu înlocuirea;
- OK - se face o singură înlocuire;
- CHANGE ALL - se înlocuiesc toate aparițiile subșirului căutat.

Opțiunea SEARCH AGAIN - permite reluarea ultimei căutări sau înlocuiri.

Opțiunea GO TO LINE NUMBER - permite deplasarea cursorului pe o anumită linie (linia și coloana în care se găsește cursorul sunt afișate în permanență).

Opțiunea FIND PROCEDURE - se deschide o fereastră în care se introduce numele unui subprogram (procedură sau funcție). Se caută acel subprogram (se caută declarația lui, și nu apelul).

Opțiunea FIND ERROR permite aflarea adresei din memorie unde s-a produs o eroare.

Un rol special în operațiunea de editare îl are o fereastră numită CLIPBOARD. Dacă nu s-a dat o opțiune specială, aceasta nu este vizibilă, dar are un rol mare și anume, în ea se pot depozita blocuri (porțiuni de text marcate într-un mod anume). La cerere, textele pot fi aduse în cadrul unei ferestre de lucru.

Dar ce este un bloc? Un bloc este o porțiune din textul sursă care este marcată (în textul sursă un bloc este vizibil într-o culoare specială). Pentru a marca un bloc, procedăm în felul următor:

- se deplasează cursorul pe prima linie care se dorește a fi marcată;
- se apasă tasta SHIFT (în mod continuu);
- parcurgem liniile din textul sursă care vor fi marcate cu ajutorul săgeților (sus sau jos).

Ce putem face cu un bloc, putem vedea analizând meniul EDIT. Acesta are următoarele opțiuni:

RESTORE LINE	
CUT	SHIFT-DEL
COPY	CTRL-INS
PASTE	SHIFT-INS
SHOW CLIPBOARD	
CLEAR	CTRL-DEL

Opțiunea CUT - permite mutarea blocului în fereastra CLIPBOARD. Mutarea se face prin ștergerea blocului din textul sursă.

Opțiunea COPY - se mută blocul în CLIPBOARD, fără a fi șters din fereastra în care lucrăm.

Opțiunea PASTE - reia blocul din CLIPBOARD și îl scrie în fereastra activă, începând din poziția curentă a cursorului.

Din cele prezentate se poate trage lesne concluzia că, în acest fel, putem muta blocuri dintr-un text în altul sau în cadrul aceluiași text.

Opțiunea SHOW CLIPBOARD - se vizualizează conținutul ferestrei CLIPBOARD.

Opțiunea CLEAR - se șterge din text blocul selectat.

Observație: La vizualizarea ferestrei CLIPBOARD se observă că este marcat ca bloc numai ultimul text salvat. Cu toate acestea, celelalte texte se păstrează. Din acest motiv, dacă acestea prezintă interes, se pot marca direct în CLIPBOARD, după regulile cunoscute.

Opțiunea RESTORE LINE - permite refacerea ultimei linii modificate (numai dacă nu am deplasat cursorul din acea linie).

Lucrul cu mai multe ferestre

De multe ori este util să lucrăm simultan cu mai multe ferestre. De exemplu, să presupunem un program care lucrează cu două unități de program. Deschidem o fereastră pentru program și câte o fereastră pentru cele două unități. Deschiderea unei noi ferestre se face tot cu F3. De la bun început trebuie precizat faptul că, dintre toate ferestrele deschise la un moment dat, numai una este activă și în ea se găsește cursorul (toate comenzile prezentate până acum se referă la fereastra activă).

În lucrul cu mai multe ferestre este util meniul WINDOW. Opțiunile acestuia se împart în două categorii:

- opțiuni privitoare la lucrul cu ferestre;
- opțiuni pentru depanare.

Aici, vom prezenta numai prima categorie de opțiuni:

SIZE/MOVE	CTRL-F5
ZOOM	F5
TILE	
CASCADE	
NEXT	F6
PREVIOUS	SHIFT-F6
CLOSE	ALT-F3

Opțiunea SIZE/MOVE - Permite modificarea dimensiunilor ferestrei active. Dacă am selectat această opțiune, fereastra curentă își schimbă culoarea. Din acest moment îi putem schimba dimensiunile. Aceasta se realizează cu ajutorul săgeților și tastei SHIFT. În mod practic, prin apăsarea tastei SHIFT și a săgeții în sus se reduce dimensiunea verticală a ferestrei. Tot așa, prin apăsarea tastei SHIFT și a săgeții stânga se reduce dimensiunea orizontală. Dacă se apasă numai săgețile, fereastra se deplasează pe ecran în sensul indicat de săgeți.

Atunci când am terminat aranjarea unei ferestre se tastează ENTER iar fereastra revine la culorile normale. În continuare se poate lucra în ea în mod obișnuit.

Opțiunea ZOOM - are rolul de a maximiza fereastra activă. Dacă există deschise mai multe ferestre acestea devin invizibile.

Opțiunea TILE - are rolul de a afișa în prim plan toate ferestrele deschise. Toate ferestrele au dimensiuni egale.

Opțiunea CASCADE - are rolul de a dispune ferestrele una peste alta (în stivă) iar fereastra activă este în prim plan.

Opțiunea NEXT - are rolul de a activa fereastra următoare celei curente.

Opțiunea PREVIOUS - activează fereastra ce a fost activă anterior celei curente.

Opțiunea CLOSE - la activarea acestei opțiuni se închide fereastra curentă. Dacă în cadrul textului s-au făcut modificări, utilizatorul este întrebat dacă se salvează sau nu conținutul.

Autodocumentarea

Se poate întâmpla să nu știm cum funcționează o anumită procedură, funcție sau să avem alte nelămuriri. Prin activarea meniului HELP sistemul integrat TURBO PASCAL ne oferă posibilitatea să ne autodocumentăm (fără a ne mai uita într-o carte sau să întrebăm pe altcineva).

În cadrul acestui meniu există următoarele opțiuni:

CONTENTS INDEX	SHIFT-F1
TOPIC SEARCH	CTRL-F1
PREVIOUS TOPIC	ALT-F1
HELP ON HELP	

Opțiunea CONTENTS - la selecția acestei opțiuni se deschide o fereastră care conține lista capitolelor sistemului HELP.

Opțiunea INDEX - la selectarea ei se afișează o fereastră care conține numele diverselor instrucțiuni, proceduri, directive de compilare etc. despre care ne putem informa. Dacă dorim informații despre un cuvânt anume (de exemplu numele unei proceduri) se apasă pe tasta SHIFT în mod continuu și se tastează cuvântul respectiv. După tastare, apare în fereastră explicația solicitată, un exemplu și, eventual, numele altor proceduri care au funcții înrudite cu procedura în cauză. Exemplele le putem copia în CLIPBOARD. Pentru aceasta, se apelează meniul EDIT cu opțiunea COPY EXAMPLE. Mai departe știm ce avem de făcut, întrucât am văzut cum se poate introduce un bloc din CLIPBOARD în fereastra curentă. Aici, putem rula acest exemplu.

Opțiunea TOPIC SEARCH - afișează aceeași fereastră ca opțiunea anterioară. Deosebirea constă în faptul că, în cazul în care cursorul se află poziționat pe un cuvânt cheie, căutarea acestuia se face automat.

Opțiunea HELP ON HELP - se oferă informații despre cum putem utiliza meniul HELP.

Depanarea programelor

De multe ori, deși credem că avem un program corect, acesta nu furnizează rezultatele cerute sau se întrerupe ca urmare a unei erori de programare. Trebuie să aflăm unde am greșit. Mecanismele puse la dispoziție ne ajută mult în acest caz (problema este cu atât mai importantă cu cât programul este mai mare).

Pentru a depana un program, avem următoarele posibilități:
- rularea pas cu pas;
- vizualizarea unor variabile;
- crearea punctelor de întrerupere.

Rularea pas cu pas

Se face prin apăsarea tastelor F8 sau F7. Diferența dintre ele este că la apăsarea tastei F7 se rulează pas cu pas și procedurile sau funcțiile apelate de program, chiar dacă acestea aparțin unor unități de program (dacă este găsită sursa acestora). Prin apăsarea tastei F8 se rulează pas cu pas doar programul principal. De remarcat este faptul că putem alterna utilizarea acestor taste în funcție de faptul dacă suntem sau nu interesați să rulăm pas cu pas o anumită procedură sau funcție. În ce constă de fapt această rulare pas cu pas? Să presupunem că am lansat programul cu ajutorul tastei F8. Apare o bară deasupra cuvântului BEGIN. Se apasă din nou F8. Bara se mută deasupra instrucțiunilor care se găsesc pe linia de după BEGIN. La o citire sau scriere, dispare textul sursă cu bară, apare partea de ecran unde programul își afișează rezultatele. După efectuarea operației se revine asupra textului. În acest fel putem vedea care sunt instrucțiunile și, mai ales, ordinea în care sunt rulate acestea. Este posibil ca o anumită secvență, pe care noi o consideram corectă, să nu fie nici măcar parcursă. *Atenție!* În cadrul acestui tip de depanare, un rol fundamental îl are dispunerea instrucțiunilor pe o linie. Să fim mai expliciți.

Considerăm două forme de dispunere a secvenței următoare:

a.
```
FOR i:=1 to n do s:=s+i;
```
b.
```
FOR i:=1 to n do
  s:=s+i;
```

În cazul primei forme, plasarea barei pe secvență se face o singură dată. În cazul formei b, plasarea barei se face de n ori pe primul rând și de n ori pe al doilea rând.

Vizualizarea conținutului anumitor variabile

Pentru a vedea ce am greșit în program este util să vizualizăm și conținutul anumitor variabile (chiar dacă acestea nu conțin un rezultat final). Cum

procedăm pentru a realiza aceasta?

De la bun început trebuie precizat că se dispune de o fereastră cu un rol special, numită WATCH. În această fereastră se vizualizează conținutul diverselor variabile care ne interesează. Pentru a realiza aceasta, se deschide fereastra WATCH cu ajutorul comenzii WATCH din cadrul meniului WINDOW. La tastarea ENTER se deschide o fereastră de dialog în care se cere numele variabilei al cărei conținut urmează să fie vizualizat în WATCH. Odată tastat acest nume, se afișează conținutul (dacă acesta este cunoscut în acel moment), în caz contrar se afișează un mesaj corespunzător iar conținutul va fi afișat imediat ce este cunoscut. De menționat este și faptul că în WATCH se poate vizualiza conținutul mai multor variabile.

Tehnica de vizualizare a conținutului anumitor variabile se poate utiliza în paralel cu tehnica rulării pas cu pas. În acest fel, avem posibilitatea de a vedea valorile intermediare ale acestor variabile.

Observație: În perioada de introducere în programare, utilizarea combinată a celor două tehnici este utilă, chiar pentru programe corect concepute. De fapt, de aici se poate înțelege cel mai bine funcționarea unui program (algoritmul). De multe ori, când explică un algoritm, profesorul reproduce tocmai această rulare pas cu pas, în care evidențiază valorile curente ale variabilelor.

Crearea punctelor de întrerupere

De multe ori suntem siguri că o primă secvență de program funcționează corect (de exemplu o secvență de introducere a datelor). Cu toate acestea, programul funcționează incorect. Ce avem de făcut? Dacă rulăm pas cu pas programul, până la secvența pe care dorim să o analizăm, pierdem timpul și răbdarea ne este pusă la grea încercare. Din acest motiv, este bine să trecem repede peste secvența de care suntem siguri.

Pentru a opri rularea programului acolo unde începe secvența pe care dorim să o testăm, trebuie să creăm un punct de întrerupere.

Cum procedăm ? Plasăm cursorul pe linia din textul sursă la care dorim să se întrerupă execuția programului. Lansăm spre execuție programul cu ajutorul tastei F4. Dacă suntem interesați, după aceasta putem muta din nou cursorul pe o nouă linie la care, când se ajunge, dorim să ne oprim. Această tehnică o putem utiliza combinat cu afișarea conținutului variabilelor. De asemenea, dintr-un punct de întrerupere se poate porni rularea pas cu pas a programului (cu tastele F7 sau F8, după cum se dorește intrarea în subprograme sau nu).

De fapt, cele prezentate până acum constituie opțiuni ale meniului RUN. Acestea sunt:

RUN	CTRL-F9
PROGRAM RESET	CTRL-F2
GO TO CURSOR	F4
TRACE INTO	F7

STEP OVER F8
PARAMETERS

Opțiunea RUN - lansează în execuție un program.

Opțiunea PROGRAM RESET

Considerăm următorul program:

```
program test;
var n:integer;
begin
  write('n=');readln(n);
  while 1<>2 do
end.
```

Acesta ciclează (nu se termină niciodată). De multe ori, din greșeală, facem astfel de programe. Trebuie să găsim o modalitate să-l oprim. Pentru aceasta se tastează CTRL + BREAK(PAUSE). Programul se întrerupe, iar pe una din liniile textului sursă apare o bară care indică instrucțiunea la care s-a oprit programul nostru. Această informație ne este de mare folos, deoarece astfel putem identifica secvența unde programul ciclează. Dacă dorim să relansăm programul în execuție (eventual cu alte date de intrare, ca să vedem dacă și în acest caz programul ciclează), constatăm că rularea programului pornește din punctul unde aceasta a fost întreruptă (lansarea a fost făcută cu CTRL + F9). Deci tentativa noastră eșuează. Avem însă posibilitatea de a porni rularea de la început, dacă tastăm, în prealabil, opțiunea PROGRAM RESET.

Crearea unui punct de întrerupere se poate face și cu ajutorul opțiunii GO TO CURSOR (în loc de F4).

Opțiunile TRACE INTO și STEP OVER au fost prezentate la rularea programului pas cu pas.

Opțiunea PARAMETERS permite introducerea unui șir care are acelasi rol ca șirul ce urmează numelui program în cazul apelului din DOS al unui program care primește anumiți parametri pe linia de intrare (vezi unitatea de program DOS).

Există și posibilitatea creării mai multor puncte de întrerupere, de la început. Pentru a realiza aceasta, se procedează astfel:

- se poziționează cursorul pe linia la care, când se ajunge, se dorește crearea primului punct;
- se apelează opțiunea TOGGLE BREAKPOINT, din meniul DEBUG, opțiune care are rolul de a marca primul punct de întrerupere (pe ecran apare o bară roșie);
- se poziționează cursorul pe următorul punct de întrerupere;
- se tastează opțiunea TOGGLE BREAKPOINT.

În acest fel se marchează toate punctele de întrerupere.
Rularea programului se face cu F4.

Există și alte opțiuni care permit depanarea programelor, însă considerăm că acestea sunt suficiente.

Compilarea programelor

Dacă se dorește numai compilare, fără lansare în execuție, se poate tasta F9. Când compilarea a decurs fără erori se afișează un mesaj corespunzător. În caz contrar, apare un mesaj de eroare iar cursorul se poziționează acolo unde se bănuiește a fi eroarea.

Observație: nu se poate identifica cu precizie locul în care a apărut eroarea. Vom înțelege motivul care duce la aceasta, în cursul anului viitor. Este sarcina programatorului să identifice cu precizie cauza erorii.

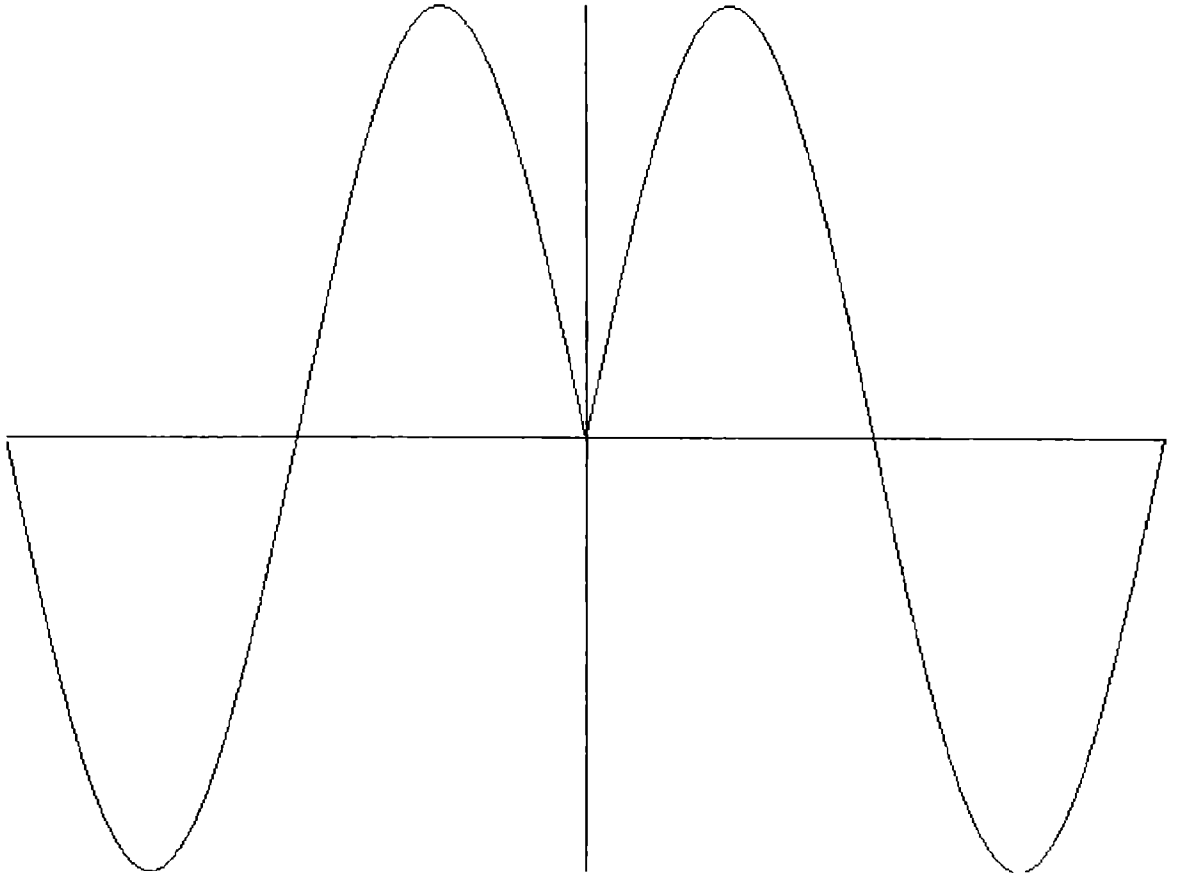
În anumite cazuri, dorim să creăm un program executabil (extensia .exe). În acest caz se procedează astfel:

- se apelează meniul COMPILE;
- se tastează D (în mod normal, un program compilat se găsește în memoria internă) care determină ca programul compilat să se găsească pe suportul magnetic.

aAbBcCdD123

aAbBcCdD123

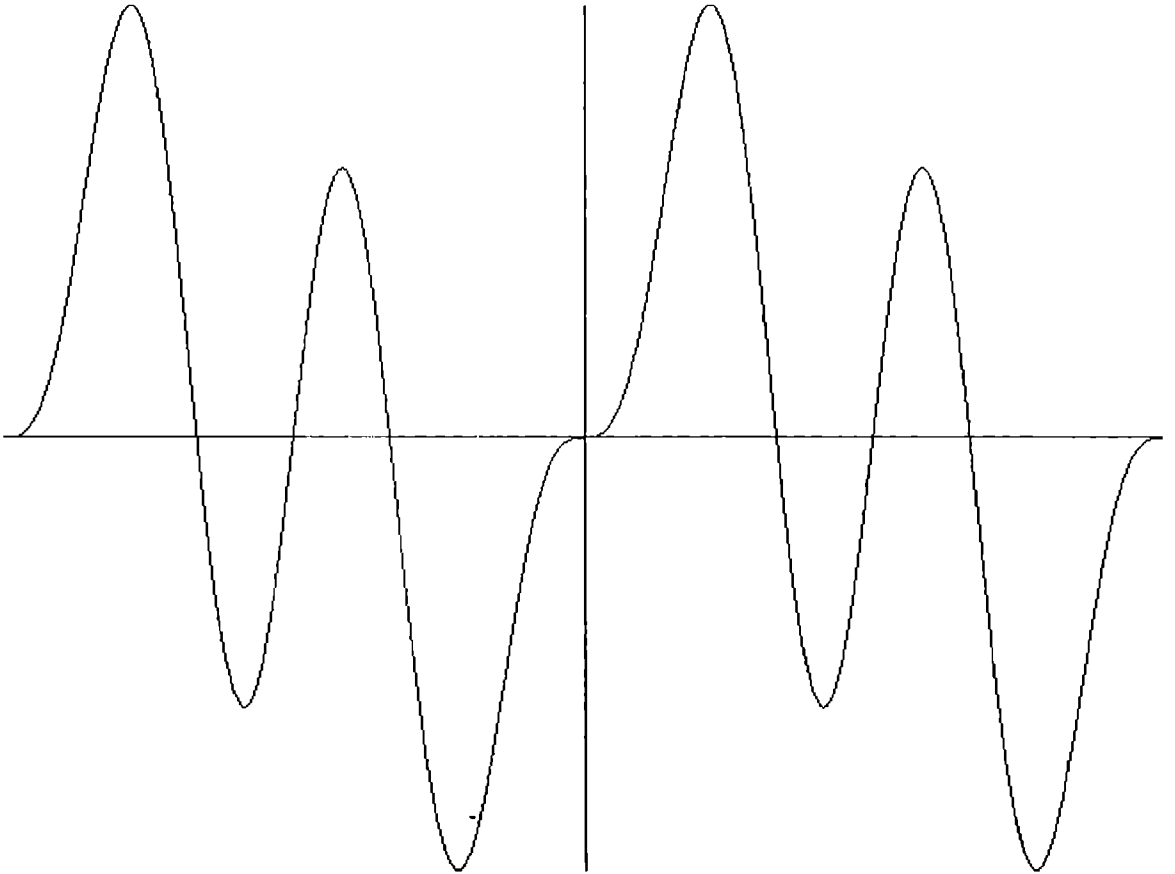
Planșa 2



f(x) = sin(abs(x))

$x \in [-2\pi, 2\pi]$

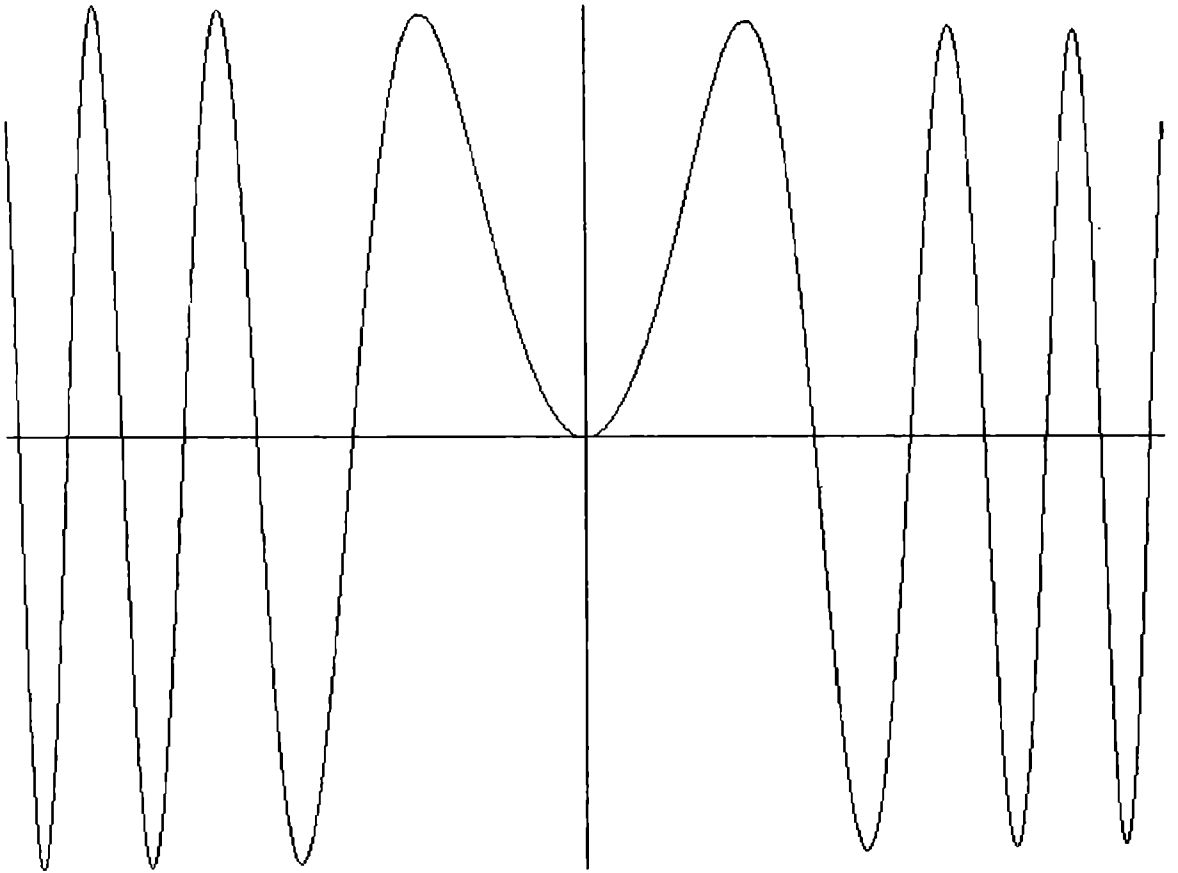
Planșa 3



$$f(x) = \sin(x) * \sin(2*x) * \sin(3*x)$$

$$x \in [-\pi, \pi]$$

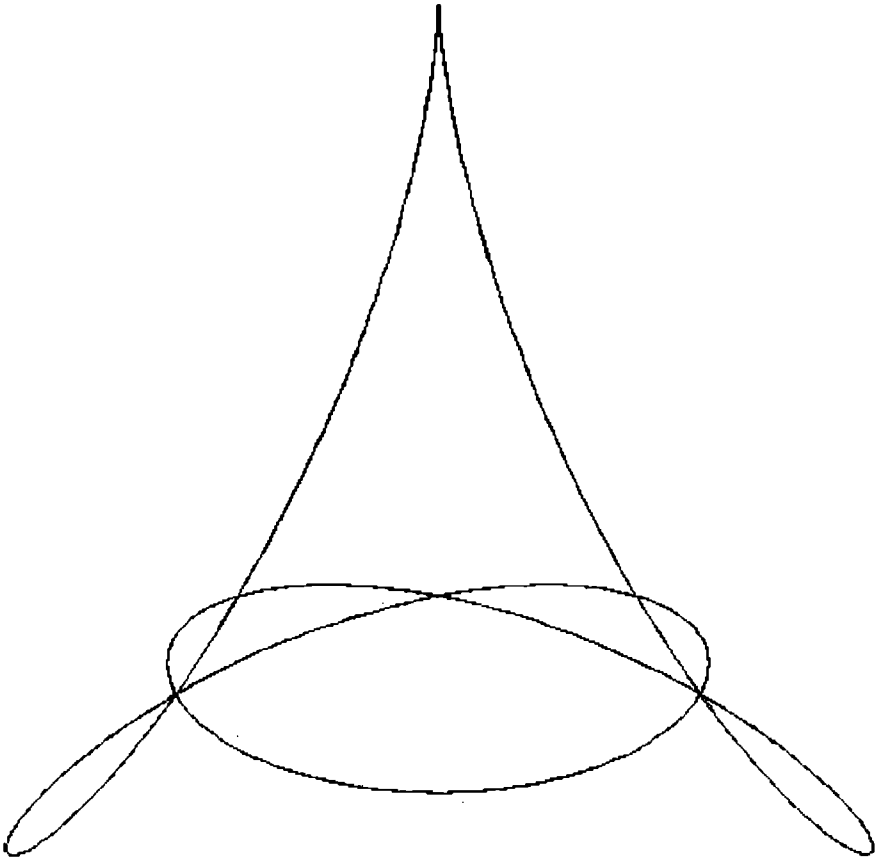
Planşa 4



$$z = \sin(x*x)*\cos(x*x)*\exp(-x/100)$$

$$x \in [-\pi, \pi]$$

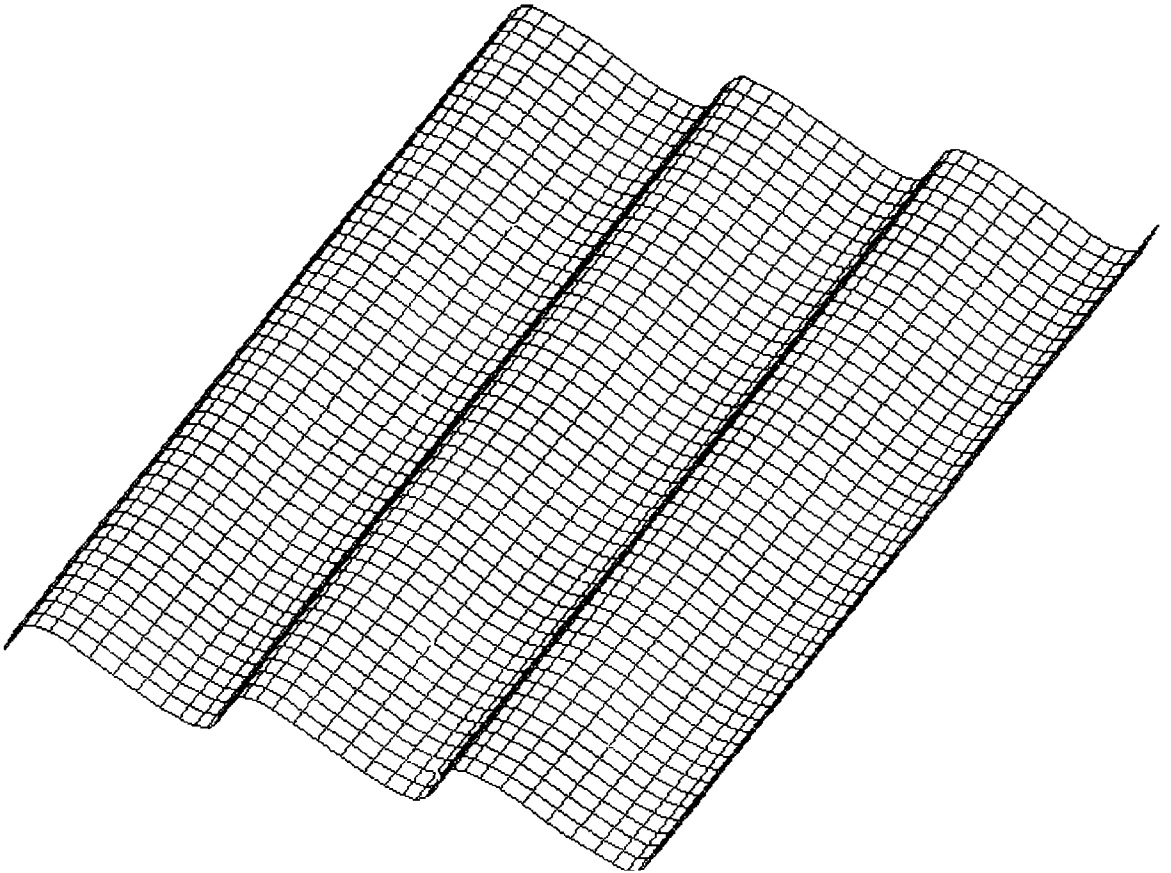
Planșa 5



$f(t) = \sin(t)*\sin(2*t)*\sin(3*t), \quad g(t) = \cos(t)*\cos(2*t)*\cos(3*t)$

$t \in [-0, 6.29]$

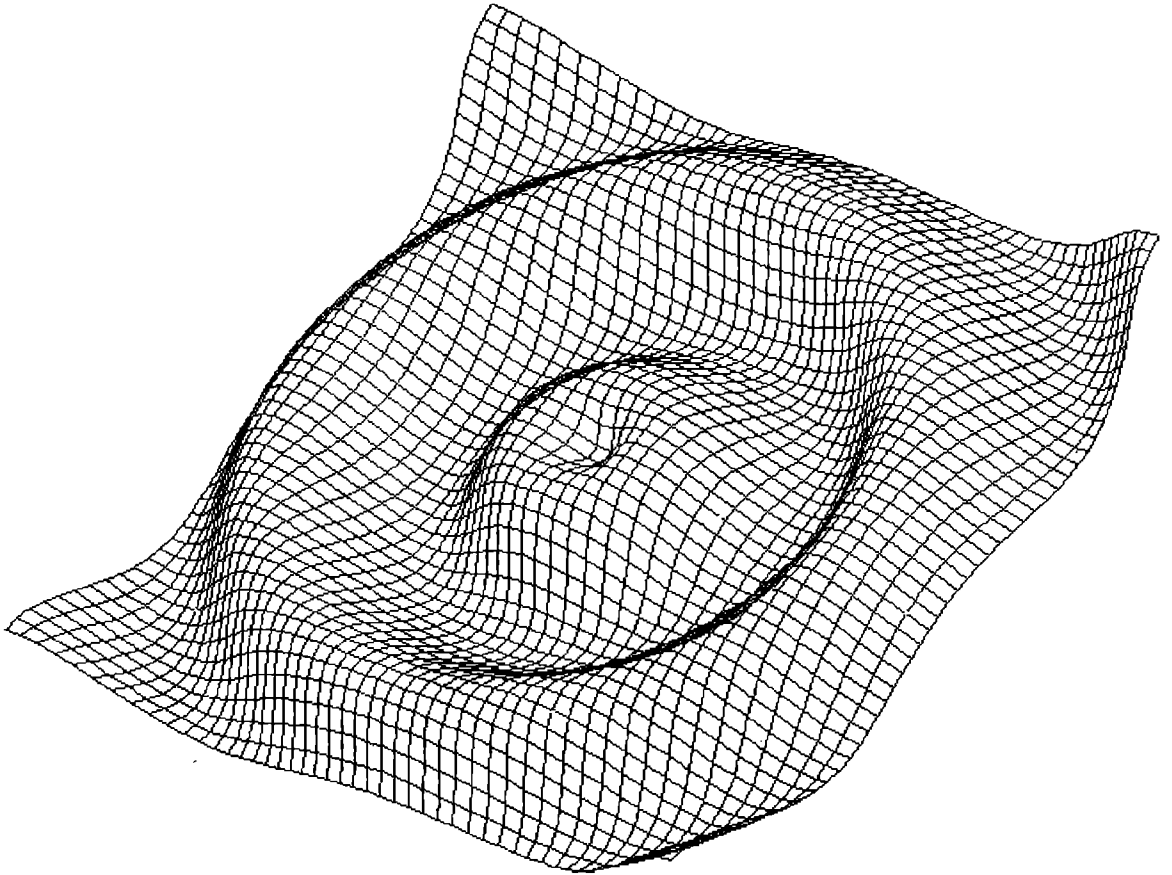
Planșa 6



z = sin(x)

x∈[-10, 10], y∈[-10, 10], $\alpha = 165$, $\beta = -135$, $\gamma = 135$

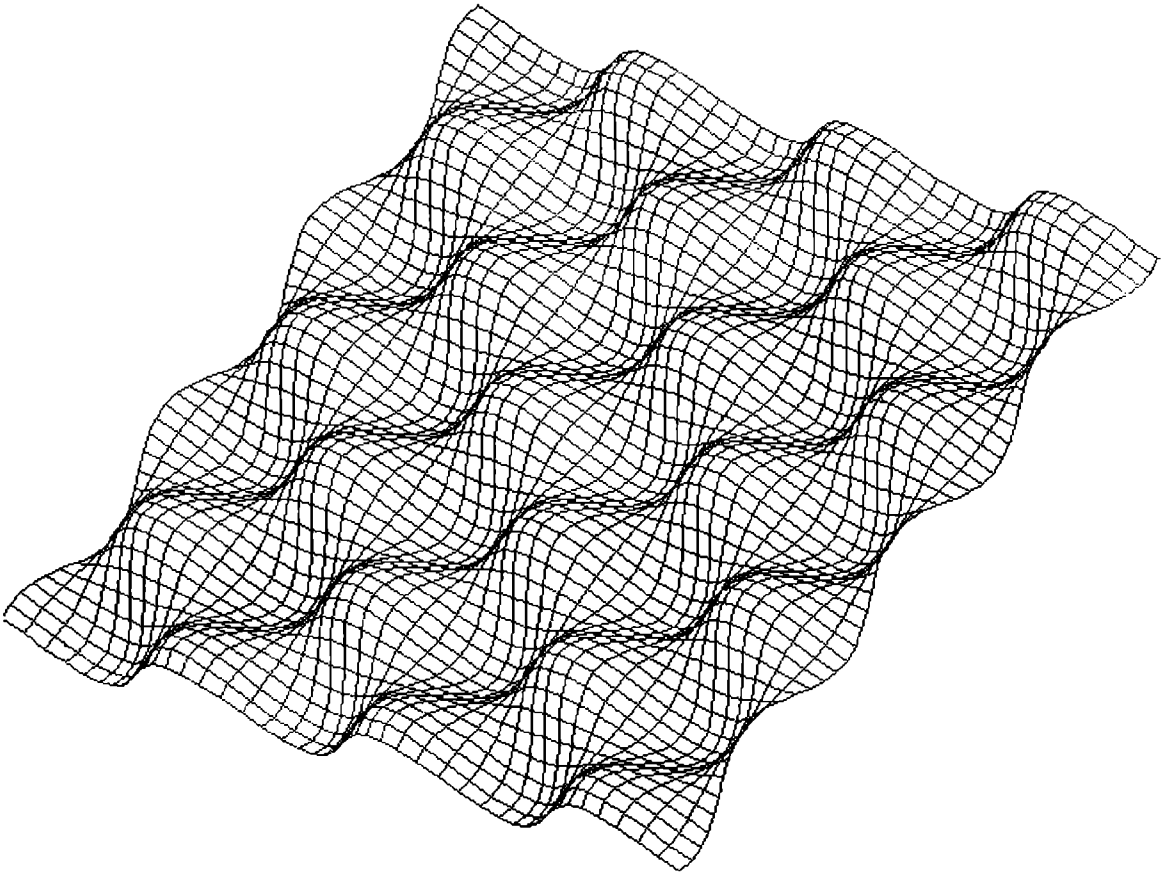
Planșa 7



z = sin(sqrt(x*x+y*y))

$x \in [-10, 10]$, $y \in [-10, 10]$, $\alpha = 165$, $\beta = -135$, $\gamma = 135$

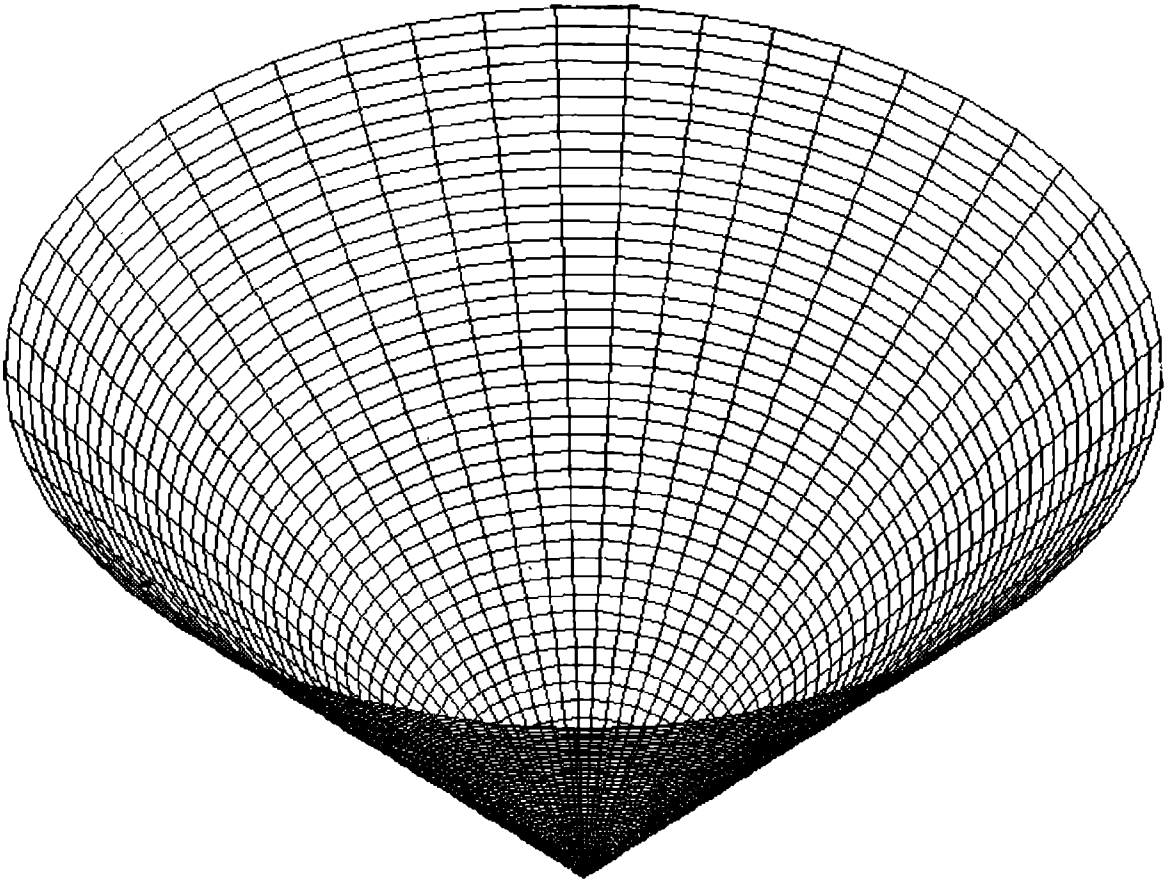
Planșa 8



$$z = \sin(x) * \cos(y)$$

$x \in [-10, 10]$, $y \in [-10, 10]$, $\alpha = 165$, $\beta = -135$, $\gamma = 135$

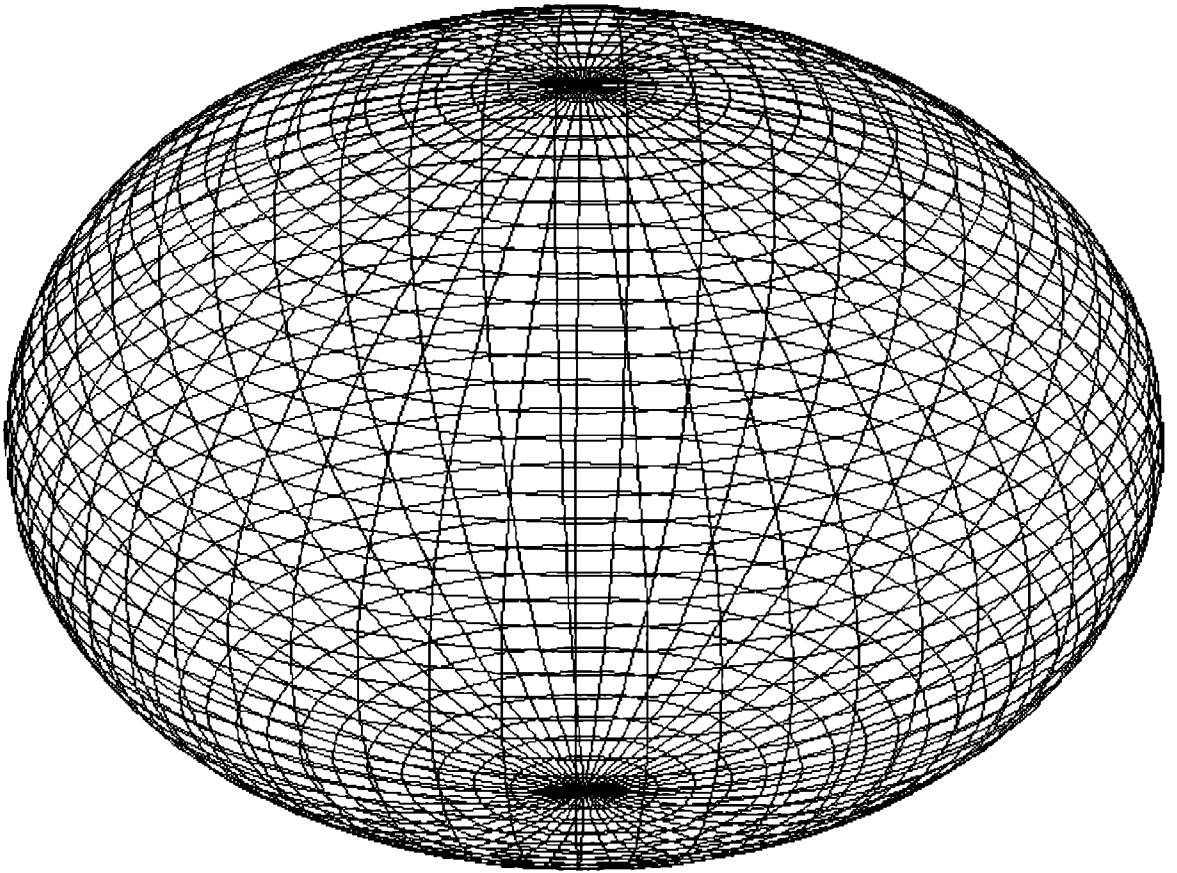
Planșa 9



f(s) = s h(s) = s

a = 0 b = 4 50 valori x 50 valori y

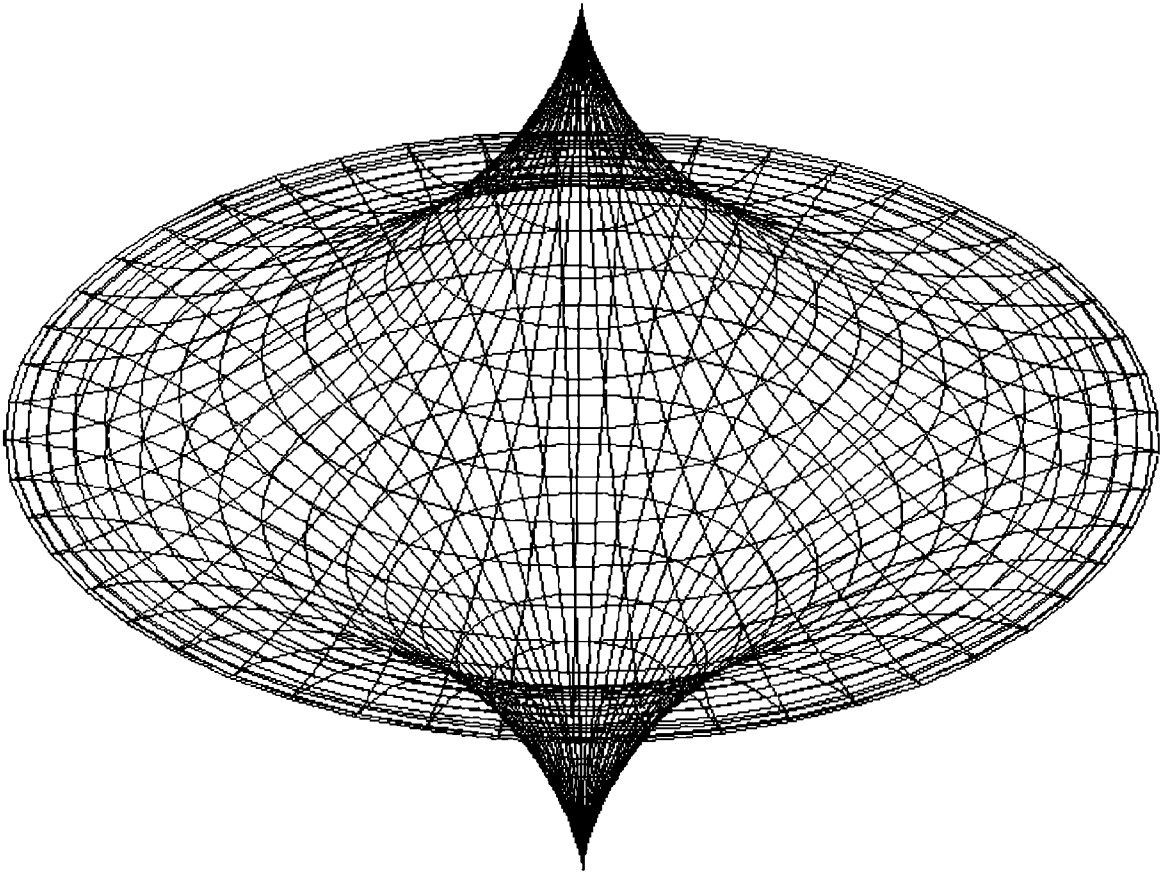
Planşa 10



f(s) = cos(s) **h(s) = sin(s)**

a = −90 b = 90 50 valori x 50 valori y

Planșa 11



f(s) = cos(s)∗cos(s)∗cos(s) **h(s) = sin(s)∗sin(s)∗sin(s)**

a=−90 b=90 50 valori x 50 valori y

Bibliografie

T. Bălănescu
Horia Georgescu
M. Gheorghe
I. Văduva
„Pascal și Turbo Pascal"
Editura Tehnică, 1992

Doina Rancea
„Limbajul Turbo Pascal"
Editura Libris, 1993

Constantin Apostol
Ioan Roșca
Bogdan Ghilic
„Prelucrarea fișierelor în Pascal"
Editura ALL, 1994

V. Cristea
& colectiv
„Inițiere în Turbo Pascal"
Editura Teora, 1995

Ion Ivan
„Structuri de date"
1992

Informatică pentru clasele IX-XII

Conceput în conformitate cu programa școlară în vigoare, pentru disciplinele **Algoritmi și limbaje de programare** și **Aplicații practice de laborator** – clasa a IX-a, profil informatică – manualul este rezultatul îndelungatei experiențe didactice a autorului.

Fiecare capitol este însoțit de un set de probleme propuse, prezentate în funcție de complexitatea lor. Elevilor li se oferă acum posibilitatea de a acoperi dintr-o singură sursă, atât programa cât și aplicații din afara ei, pentru care există un interes dovedit: unități de program, programare pe obiecte, crearea meniurilor, formatarea ecranelor, validarea datelor de intrare, reprezentări grafice de funcții și suprafețe, salvarea imaginilor etc.

Rețineți culorile seriilor de cărți pentru elevi:

ROȘU pentru clasele I-IV

ALBASTRU pentru clasele V-VIII

GALBEN pentru clasele IX-XII

Editura Teora ISBN 973-601-212-3 Lei 8000